(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A262 153

DTIC
S ELECTE D
APR 1 1993
C

THESIS

EFFICIENT GRID BASED TECHNIQUES FOR SOLVING THE WEIGHTED REGION LEAST COST PATH PROBLEM ON MULTICOMPUTERS

by

Cengiz EKIN

December 1992

Thesis Advisor: Amr ZAKY

Approved for public release; distribution is unlimited.

93 3 31 133

93-06694

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release;
distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Computer Science Dept.
Naval Postgraduate School

6b. OFFICE SYMBOL *(if applicable)*
CS

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State, and ZIP Code)*
Monterey, CA 93943-5000

7b. ADDRESS *(City, State, and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(if applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO

11. TITLE *(Include Security Classification)*
EFFICIENT GRID BASED TECHNIQUES FOR SOLVING THE WEGHTED REGION LEAST COST PATH PROBLEM ON MULTICOMPUTERS (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
EKIN, Cengiz

13a. TYPE OF REPORT
Master's Thesis

13b. TIME COVERED
FROM TO

14. DATE OF REPORT *(Year, Month, Day)*
December 1992

15. PAGE COUNT
100

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Numerical Path Planning, Weigted Regions, Grid Based Techniques, Transputers, Distributed Computing, Parallel Processing,

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This thesis explores the possibilities of developing fast grid-based parallel algorithms to solve the Weighted Region Least Cost Path problem. Two complementary steps have been undertaken. First, an efficient sequential algorithm to solve the above problem was developed. The algorithm is a modification of a Gauss-Seidel-like algorithm for obtaining the minimum costs. The most salient feature of the algorithm is the reduction of the number of nodes and edges in cheaper regions of the grid. The reported experimental results ascertain the superiority of this algorithm with regard to computer running time at a modest reduction in the accuracy of the obtained solution. Parallel implementations of grid-based algorithms were studied. A simple grid-based variant was implemented on a network of Transputers. The overall approach employed could be used to develop a parallel version of the above sequential algorithm on a Transputer network, combining both advantages of efficiency and parallelism.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Amr ZAKY

22b. TELEPHONE *(Include Area Code)*
(408) 646-2693

22c. OFFICE SYMBOL
CS/Za

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ABSTRACT

This thesis explores the possibilities of developing fast grid-based parallel algorithms to solve the Weighted Region Least Cost Path problem. Two complementary steps have been undertaken. First, an efficient sequential algorithm to solve the above problem was developed. The algorithm is a modification of a Gauss-Seidel-like algorithm for obtaining the minimum costs. The most salient feature of the algorithm is the reduction of the number of nodes and edges in cheaper regions of the grid. The reported experimental results ascertain the superiority of this algorithm with regard to computer running time at a modest reduction in the accuracy of the obtained solution. Parallel implementations of grid-based algorithms were studied. A simple grid-based variant was implemented on a network of Transputers. The overall approach employed could be used to develop a parallel version of the above sequential algorithm on a Transputer network, combining both advantages of efficiency and parallelism.

DTIC QUALITY INSPECTED 4

Accesion For
NTIS CRA&I ☑
DTIC TAB □
Unannounced □
Justification ______

By ______
Distribution /

Availability Codes

Dist | Avail and / or Special
A-1

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. GENERAL

Numerical path planning has been studied quite a bit in the last few years. One problem of numerical path planning involves finding the optimal path from a given starting point to a goal point through a plane that has been subdivided into weighted regions. This prob.em is known as the Weighted Region Least Cost Path (WRLCP) problem. The best path can be minimizing some cost (e.g. energy or time).

Several approaches can be used to tackle this WRLCP problem. Some of these approaches and their pros and cons are presented in chapter II.

One of the most significant constraints has been the ability to evaluate in real time the optimal path through a weighted region. Traditional sequential algorithms can quickly become overloaded with the sheer number of calculations required for a realistic problem. Consequently, several algorithms for solving this type of problem have been researched in recent years. These algorithms have, to a varying degree, potential for parallelization. Parallel algorithms have a tremendous advantage in speed, but also may open the door to new problems, especially in mapping, scheduling and coordinating the different parallel activities. Multicomputer networks are no exception. The principal goal in parallelizing an algorithm is to sustain a close-to-linear speedup in processing time. This is no trivial task, since the processor communication reduces the speedup. This, and the fact that parallelizing an algorithm may increase its overall processing time (summed over the total network of processors) make this problem challenging.

We investigated in this thesis some of the basic approaches with their potential for parallelization, and implemented one of these algorithms, first sequentially and then in parallel on a Transputer network.

B. THESIS OUTLINE

Chapter II introduces an analysis of the different parallel path approaches which have been investigated recently. The algorithm which appears most promising for solving the weighted region least cost path problem, which is a grid algorithm, has been selected for further study. In chapter III, implementation and some modifications of this algorithm is presented with an illustrative example. Chapter IV provides a description of the Transputer network and ADA. In chapter V, this approach has been developed into a parallel algorithm and implemented on an INMOS Transputer network using ADA. The algorithms ability to use parallel computing to solve arbitrary WRLCP problems has been investigated. Finally, these results will be compared with the results of single processor, and with previous results on a multicomputer network. Conclusions and recommendations for further research are offered in chapter VI.

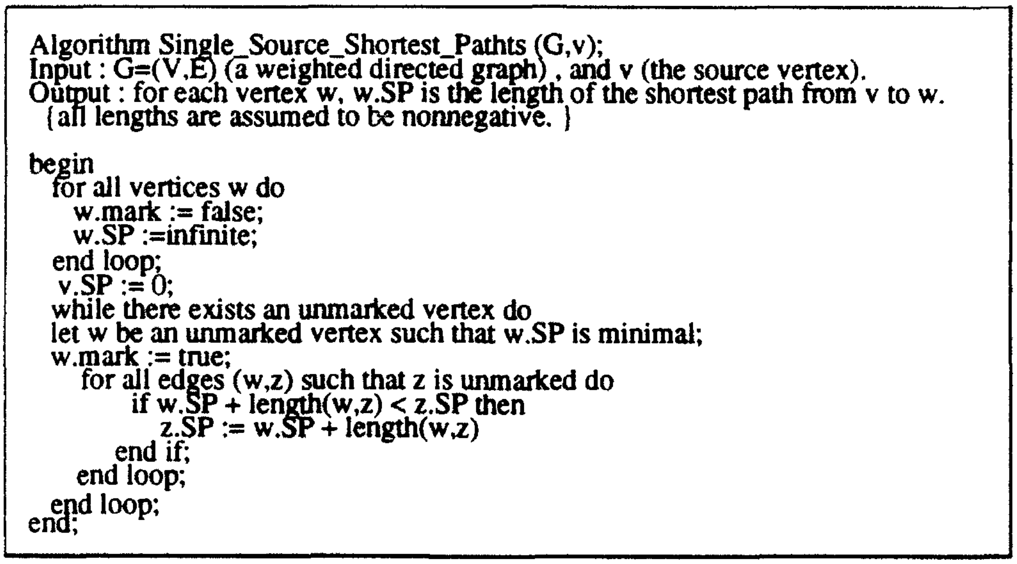
II. BACKGROUND

We overview in this chapter some of the basic approaches for solving the WRLCP problem and we investigate their potential for parallelization.

A. SINGLE SOURCE SHORTEST PATHS (SSSP)

1. Dijkstra's Algorithm

The algorithm for SSSP presented in Figure 1, which was developed by Dijkstra in 1959, is the starting point for this problem [MAN89].

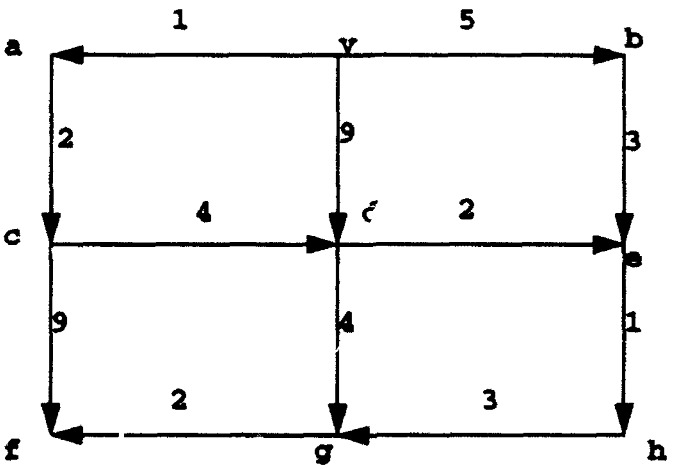


```
Algorithm Single_Source_Shortest_Pathts (G,v);
Input : G=(V,E) (a weighted directed graph) , and v (the source vertex).
Output : for each vertex w, w.SP is the length of the shortest path from v to w.
 {all lengths are assumed to be nonnegative. }

begin
  for all vertices w do
    w.mark := false;
    w.SP :=infinite;
  end loop;
  v.SP := 0;
  while there exists an unmarked vertex do
  let w be an unmarked vertex such that w.SP is minimal;
  w.mark := true;
     for all edges (w,z) such that z is unmarked do
          if w.SP + length(w,z) < z.SP then
             z.SP := w.SP + length(w,z)
        end if;
    end loop;
  end loop;
end;
```

Figure 1: Dijkstra's Algorithm

In Figure 2, a small example is presented to demonstrate the algorithm. The first line includes only paths of one edge from *v* (the source). The shortest path is chosen, in this case, leading to vertex *a*. The second line shows the update of the paths including now all paths of length one from either *v* or *a*, and the shortest path now leads to *c*. A new vertex is added in each line, and the current known shortest paths from *v* are listed to every vertex. The underlined distances are those that are known to be the shortest. The algorithm keeps adding new vertices to the selected list until all vertices are added.



	v	a	b	c	d	e	f	g	h
a	0	1	5	∞	9	∞	∞	∞	∞
c	0	**1**	5	3	9	∞	∞	∞	∞
b	0	**1**	5	**3**	7	∞	12	∞	∞
d	0	**1**	**5**	**3**	7	8	12	∞	∞
e	0	**1**	**5**	**3**	**7**	8	12	11	∞
h	0	**1**	**5**	**3**	**7**	**8**	12	11	9
g	0	**1**	**5**	**3**	**7**	**8**	12	11	**9**
f	0	**1**	**5**	**3**	**7**	**8**	12	**11**	**9**

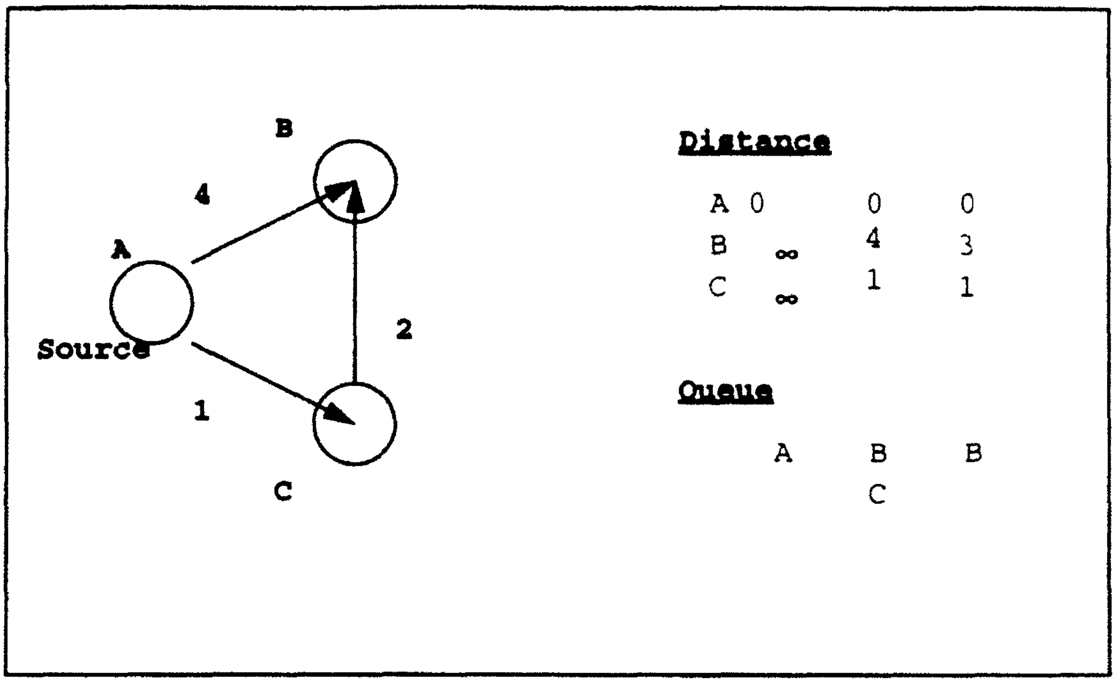
Figure 2: An Example Of The SSSP Algorithm

The algorithm can be easily extended from directed to undirected form.

2. Parallelization Of SSSP

Dijkstra's algorithm has been parallelized by Moore [QUI87]. He devised two parallel algorithms. The first algorithm makes the **for** loop (in Figure 1) parallel, which explores the outgoing edges from a given vertex. the second method is to parallelize the **while** loop (in Figure 1); that is, at any one time in the execution of the algorithm there are probably many vertices in the queue. The parallelizability of the first method is restricted by the number of edges outgoing from each vertex. On the other hand, the second method performs larger tasks, that produces good speedup. More detailed information can be found in the given reference.

In the parallel algorithm based on the second method, a queue is used. That queue is initialized with the source point, and then a number of asynchronous processes are created. Each of these processes goes through the steps of deleting a node from the queue, examining its outgoing edges, and inserting into the queue the nodes to which shorter paths have been found. In Figure 3 an example is presented in which the number of nodes in the queue shows actually the number of processes that can be parallelized. Distances are kept until they are reached from another direction. When there are more than one edge reaching to a node, minimum cost is chosen.



Distance

A	0	0	0
B	∞	4	3
C	∞	1	1

Queue

A	B	B
	C	

Figure 3: Parallelized SSSP Algorithm Example

B. RAY TRACING

1. Snell's Law of Refraction

Snell's law [HEC87] defines the path a ray of light takes as it passes from one medium to another. The ray is refracted across the border between the media according to the following equation,

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$

where n_1, n_2 are the indices of refraction and the angles are of incidence and refraction respectively (Figure 4). Fermat's principle which implies Snell's law states that light follows a path between two points such that it takes the minimum time. It can be proven (for example see [RIC87]), that a similar principle govern the path a particle takes across two regions, in which the speed of the particle is uniform in each region.

Therefore, we can apply the principles of optics to solve the WRLCP problem by assuming regions as optical media and weights as indices of refraction.

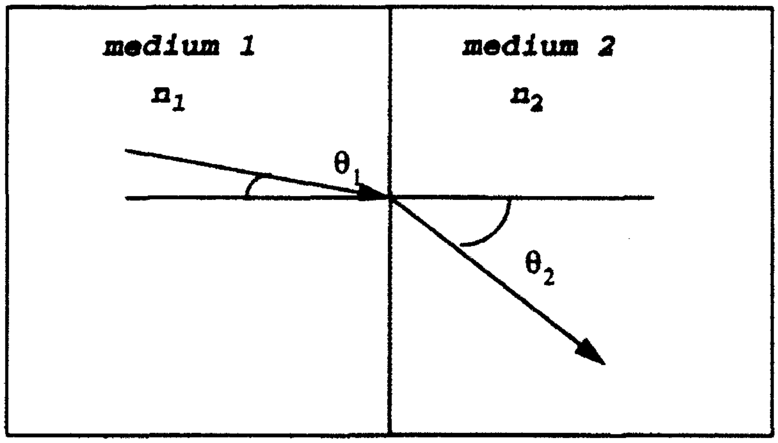


Figure 4: Snell's Law Of Refraction

2. Implementation

The main step of this implementation is to shoot a ray from the source and see where it is going to land. We keep doing this for initial rays with different angles until some of the rays hit the goal. In essence, the boundaries borde the weighted regions, the index of refraction depends on the weights and the rays refract on the border to border until it reaches the goal. At every boundary of two regions the ray obeys Snell's law of refraction. Among those rays that hit the goal point we obtain the WRLCP. In Figure 5, at the source point the short rays are just to give an idea about different set of angles and continuing three rays are given to make a clear example.

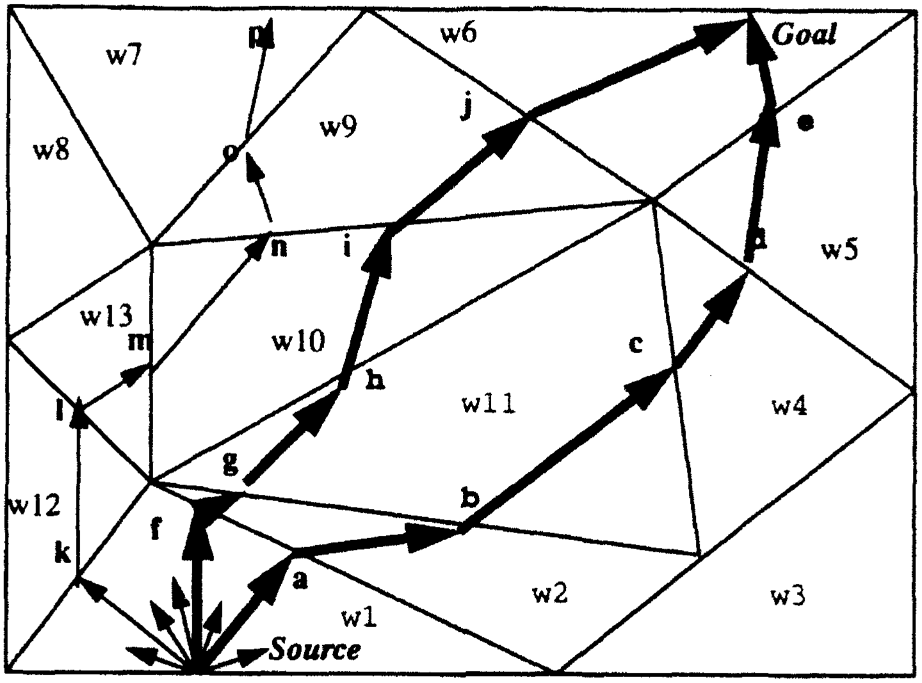


Figure 5: Ray Tracing

Through different weighted regions, as the path **Sklmnop** misses the goal, the minimal paths **SabcdeG** and **SfghijG** hit it, and one of them is chosen as WRLCP after comparison.

3. Parallelization

The Snell's law based algorithm can be easily parallelized by assigning a different set of angles to each processor. If a solution is found, it is optimal. The cost of finding the intersection point of a ray with a triangle is not large. However, the algorithm suffers several drawbacks: The ray inversion, possible presence of blind regions, and use of expensive trigonometric functions. More details can be found in [RIC87].

C. EDGE SLICING

1. Implementation

In this technique, every edge is divided into a number of segments of equal length. Some points are placed equidistantly on every map edge in the triangulated plane. A graph is constructed by connecting every two points on two edges belonging in the same triangle. The distance between two consecutive points is made proportional to some function of the costs of the two regions separated by that edge. A graph is constructed whose nodes are the original triangles' vertices plus the points which divides the edges into segments. The edges of this graph are the original triangles' edges plus the lines connecting every two non-colinear points in the same graph. In Figure 6, edge slicing is shown for the case where an edge is divided into two equal segments.

2. Parallelization

After the edges in a graph are sliced, the WRLCP problem is reduced to a SSSP problem, and the technique used for parallelizing SSSP can be invoked. A variant of this algorithm is shown in [KIN91].

D. GRID ALGORITHMS

Another approach for approximating the WRLCP problem is to model the terrain as a grid. Simply, a grid is laid over the terrain. The map is divided into equidistant grid points. The weights in the regions are transferred as edge costs. Figure 7 illustrates how a node is connected to different number of neighbors (we implemented a node with 4-neighbors, but it is easy to extend the implementation for more neighbors).

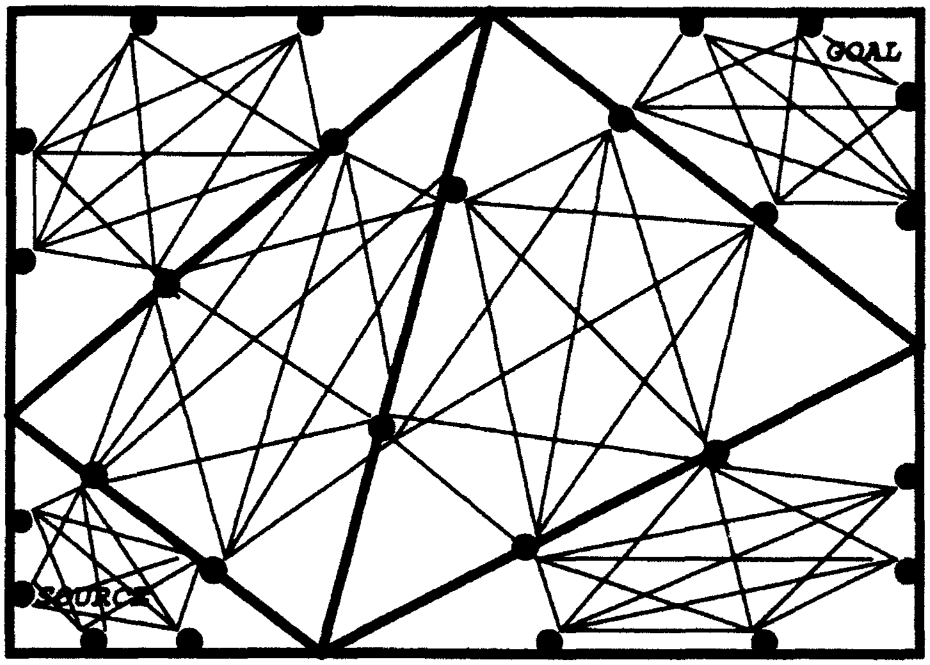


Figure 6: Edge Slicing (2-Point)

Finally, the shortest path analysis is performed on the graph. Two classes of grid algorithms could be used. These are explained below.

1. The Wavefront Variant

Starting with the source point at time zero, it progresses one step every time unit in all directions adding the appropriate nodes (these nodes which can be reached in the earliest at this time step) to a wavefront depending on the edge weights and direction of propagation. The wavefront keeps advancing every time unit until it hits the goal point. It is obvious that the progress in the inexpensive areas is bigger than expensive ones, since the length of step is proportional

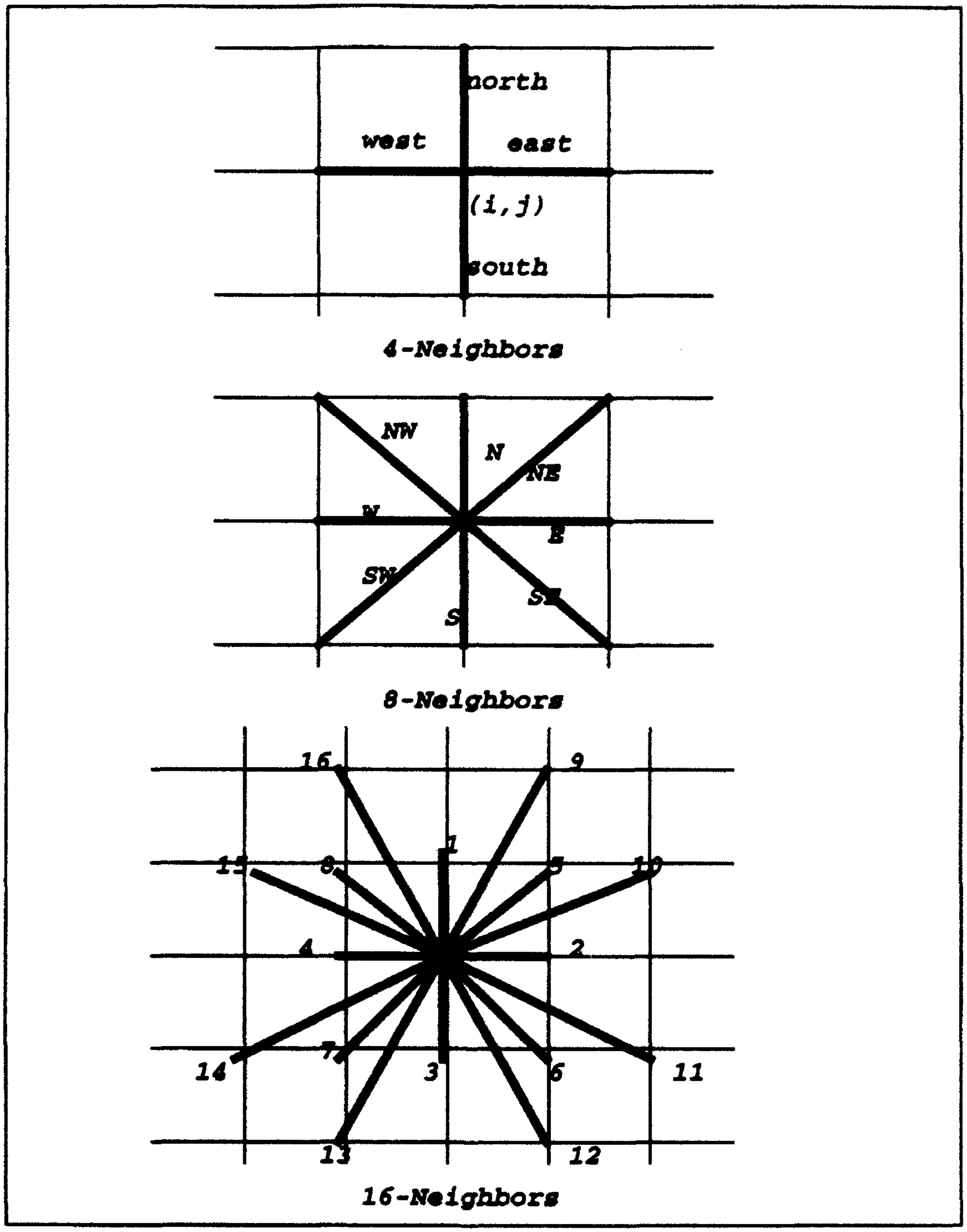


Figure 7: Neighbors In Grid Algorithm to cost and time. Figure 8 shows an example in which the bigger strides take place in the low cost areas.

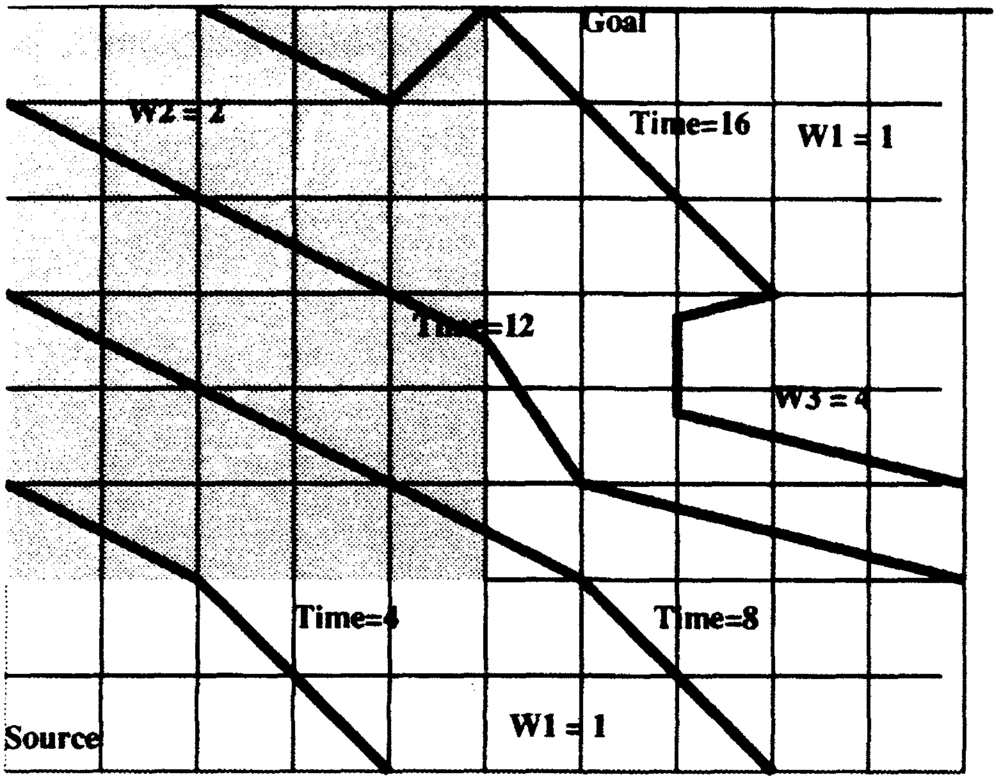


Figure 8: Wavefront Technique

One wavefront sweep will find the shortest path. But such shortest path may need a large number of time steps depending on edge weights. The number of nodes on the wavefront to be processed at every point in time varies; that causes difficulty in scheduling them on a distributed computer.

2. Relaxation-Based Approach

The reason why this approach is called "relaxation-based" is because it is similar to *Gauss-Seidel iteration* class of computation in *Partial Differential Equations* [HAB87].

$$c_{i,j}(m+1) = c_{i.j}(m) + \omega[c_{(i-1).j}(m) + c_{(i+1).j}(m+1)$$

$$+ c_{i.(j-1)}(m) + c_{i.(j+1)}(m+1) - 4c_{i.j}(m)]$$

where $c_{i,j}(m)$ shows value of grid point in row i, column j of step m.

The bracketed term indicates the change that occurs after each iteration as $c_{i,j}(m)$ is updated to $c_{i,j}$ $(m+1)$. Similarly, in our implementation, as shown in Figure 9, in each iteration, a node is updated. The cost of node is minimized by comparing it with the neighbors according to the following pseudocode:

```
$c_{current}(i,j)$ = min [ $c_{old}(i,j)$, $c(i,j+1)$ + north_weight
                                   $c(i+1,j)$ + east_weight
                                   $c(i,j-1)$ + south_weight
                                   $c(i-1,j)$ + west_weight ]
```

We decided to implement the relaxation-based grid algorithm because of the straight-forward solutions on a distributed computer.

a. Implementation

The grid graph is represented as a two-dimensional array of records. Data structure and information carried by every node is shown in Figure 10 and the variables are described below.

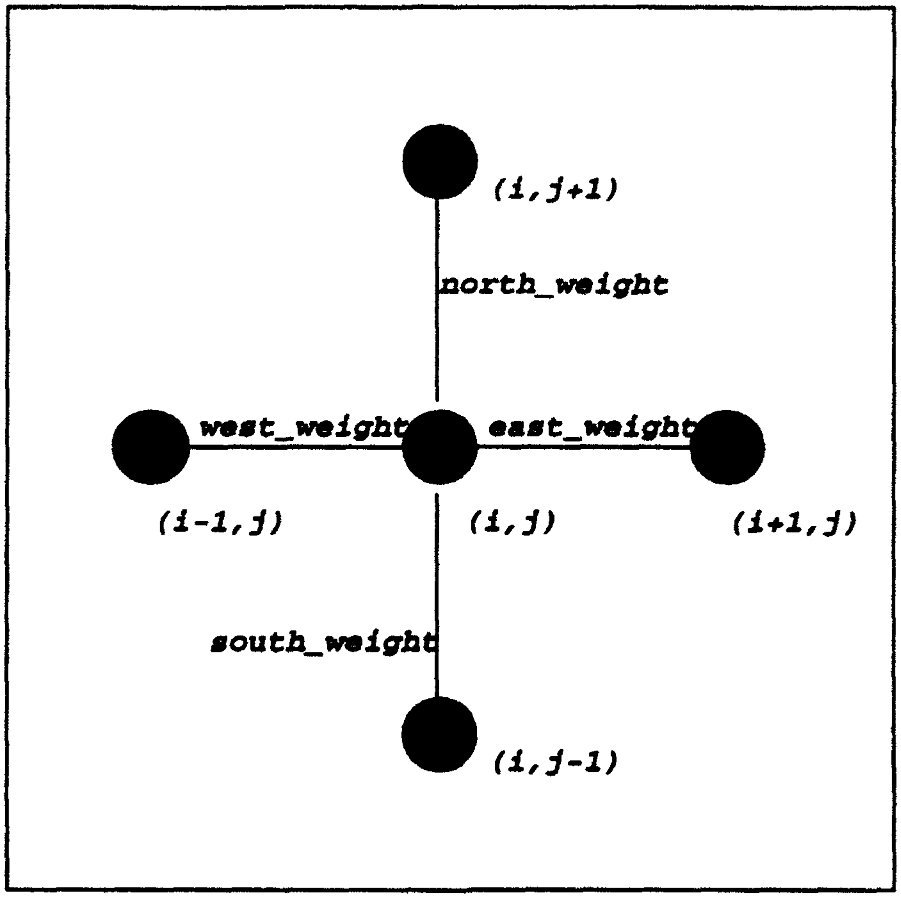


Figure 9: A Node Described With Its Neighbors And Their Relations

Current_cost: The beginning and updated cost value of the node.

Old_cost: Old_cost is the previous iteration cost value of the node and used for determining the change in cost.

N,E,S,W: North, East, South and West neighbors.

Weight: The weight on edge toward that direction.

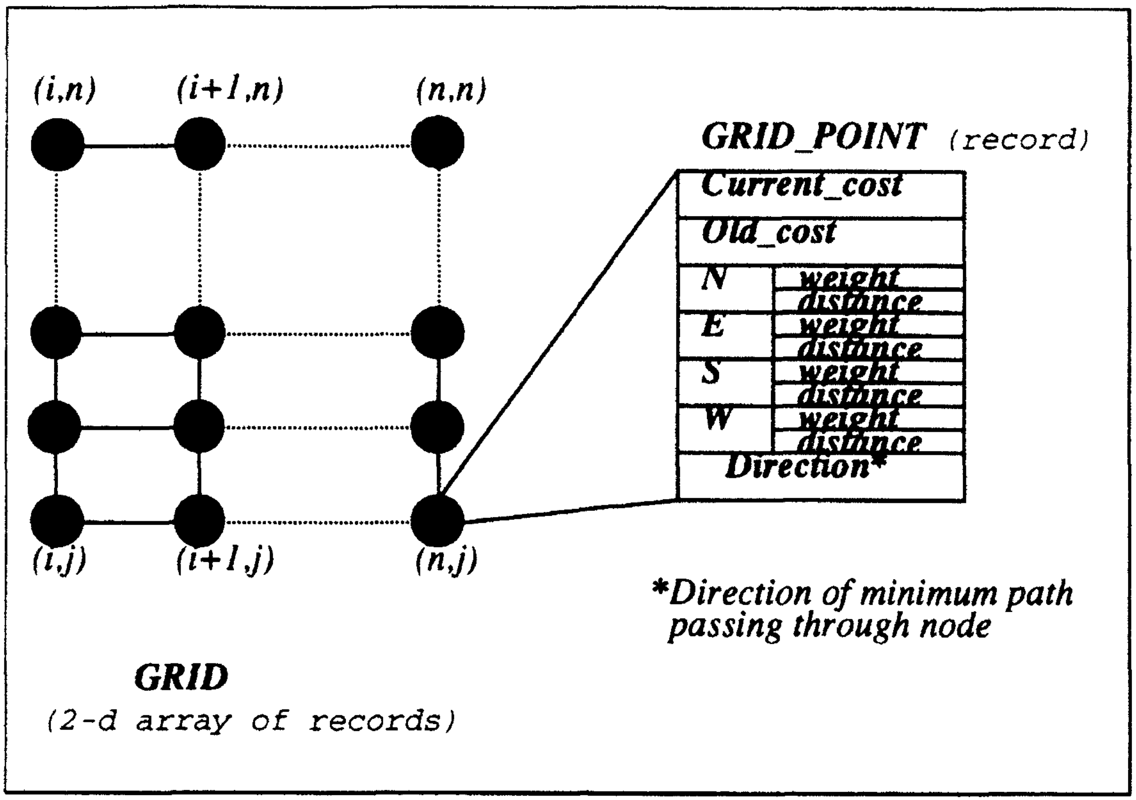


Figure 10: Data Structure

Distance: The step between two nodes (here it is always equal to one, but it changes in the modified algorithm).

Direction: String("North","East","South","West") and used for showing the path in the output.

The pseudocode for the algorithm is presented in Figure 11.

We obtained a data file which has 6400 raw data values representing the heights of 80*80 grid approximation of region. We constructed costs for the edges of the grids based on some function of these heights. For border nodes,

```
Algorithm: To find the Weighted Region Least Cost Path

Given: Two dimensional grid of points· GRID
       Source point: (is,js)
       Goal point: (ig,jg)
       Threshold: T

Output: Cost of minimization from source to goal: Cost
        The minimum path from source to goal:
        (is,js).....(ig,jg)

procedure:

 Initialize
 Read the data file
 for all nodes loop
  calculate weight(i,j)
 end loop
 xx --At this point code will be inserted in the
    --modified version of the algorithm (see Fig.14)
 flag = true
 while flag loop
    flag = false
    for all nodes loop
     find minimum value
     change = node(i,j).old_cost-node(i,j).current_cost
     if change > T then
         raise flag
      end if
     end loop
 end loop
 output the WRLCP
```

Figure 11: The Psuedocode For Relaxation-Based Algorithm

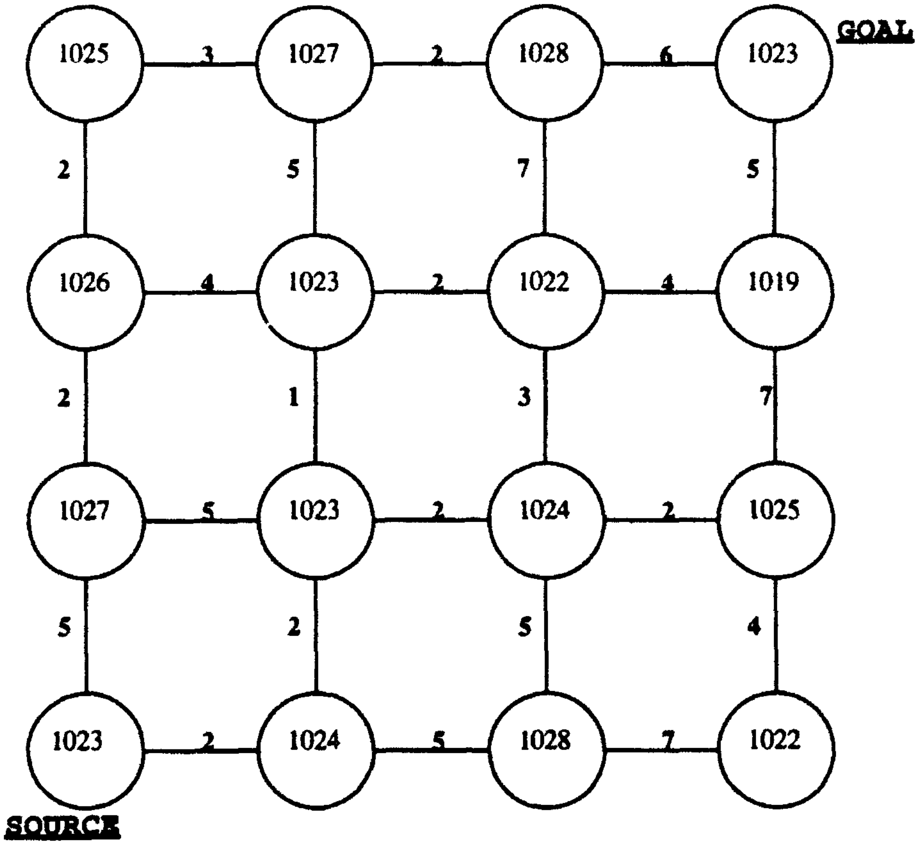
edges leading out of the grid where assigned very high positive values for weights, and minus one to distance. For all nodes we initialize the costs to a maximum positive number in order to use them in comparisons for finding the minimum.

To output the Least Cost Path, we utilize a stack since the path traces back from the goal to the source point. So the output is displayed from source to goal as cost, direction and distance (distance is equal to one in the straight-forward algorithm) are calculated at each step.

A small-sized (4*4) example is presented in the Figure 12 for clear understanding.

b. Parallelization

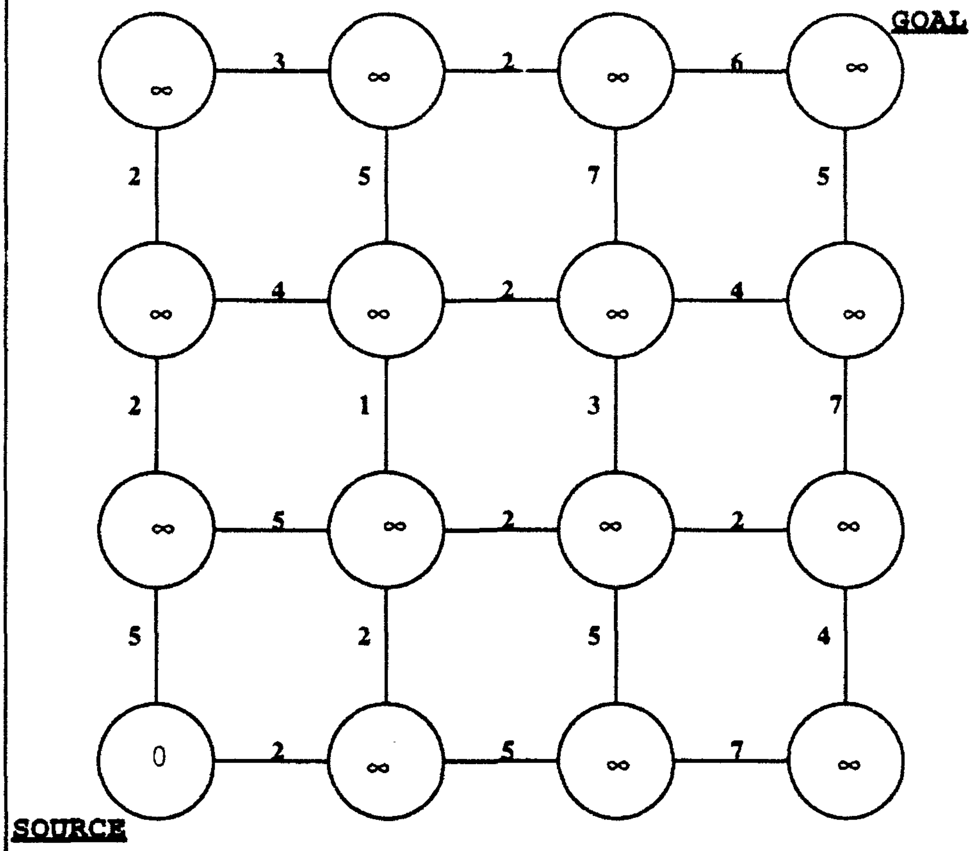
In the relaxation-based approach, solutions can be straightforward on a vector computer or a on 2-d mesh of processors. But also there are some unattractive features such as the algorithm is intrinsically nonoptimal (that is true for the wavefront technique, too). These algorithms do not reward the areas with the low cost by reducing the computations there. If the relaxation approach is used on a realistic number of processors, it is not known how to partition the computations. For instance, only one of the processors has the source point and starts computing, the computations of the rest of the processors for taking the minimum of infinite values are useless until they get the border values from that processor.



Step 1. The heights are read from the file and loaded to nodes.

Step 2. The weights on the edges are calculated using some functions (e.g. abs(1024 - 1023) + 1 = 2)

Figure 12.a: Example For Relaxation-Based Algorithm The Values Inside The Nodes Are The Heights

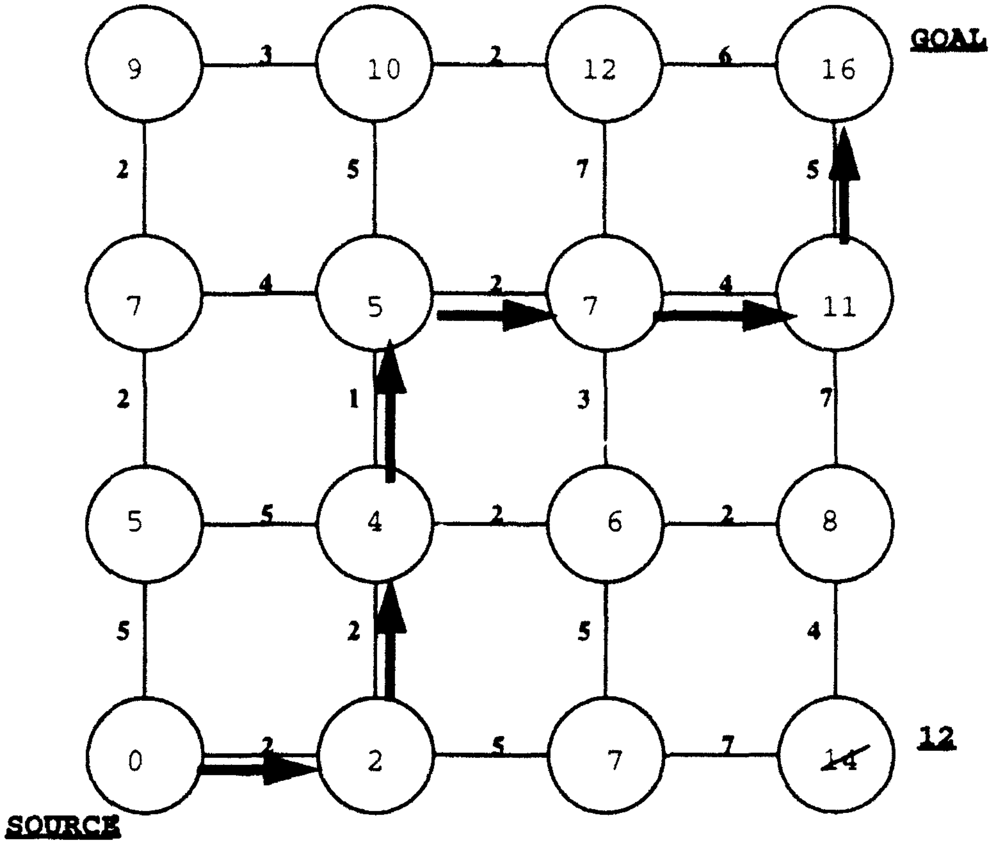


(continued from Figure 12.a)

Step 3. Source point.current_cost = 0

The others.current_cost = ∞

Figure 12.b: Example For Relaxation-Based Algorithm
The Values In The Nodes Are The Initial
Costs From The Source.



(continued from Figure 12.b)

Step 4. This is the first iteration. In the latter iterations continue.

Step 5. 14 becomes 12 after first iteration of *cost minimization*.

Step 6. Thick arrows show the minimum least cost path

Figure 12.c: Example For Relaxation-Based Algorithm The Values Inside The Nodes Are The Minimum Costs From The Source

III. A MODIFIED GRID ALGORITHM

A. INTRODUCTION

The complexity of the straight-forward grid algorithm in worst-case is $O(n^4)$ for an *n* x *n* grid. In the worst case, the WRLCP passes through all the nodes. Since the algorithm does not take the advantage of the low cost regions, we modified the algorithm to remedy this defficiency.

B. VARIABLE GRID SIZE APPROACH

We can use relaxation to obtain the shortest path from the source to different points without updating the nearest neihgbors as in the classical technique. We update the farthest neihgbors (two horizontal and two vertical in case of a 4-neihgbor technique) whose weighted distance is less than or equal to a prespecified constant. Determining such neighbors is done at the initialization phase. Rather than having the wavefront advance by a fixed distance, this algorithm has it advance by a fixed weight, thus bigger strides are made in cheaper regions.

Information that decides which neighbor to propagate to is shown in Figure 13. It assumes that every grid point will have four records (N,E,S,W), and every record will have two components: the number of grid points to the neighbor to be updated (one in the classical method), and the weighted distance to the neighbor.

Algorithm: I.To make horizantal search and elimination
II.To decide which neghbor to propagate

Given: Two dimensional grid of points: GRID (all edges calculated)
Source point: (is,js)
Goal point: (ig,jg)
Stride

Output: A searched grid and neighbor for node(i,j) to propogate

procedure horizantal search and elimination

```
 for all rows i loop
  j = 1
  while j < jmax loop
    j = $j_0$
    loop
      if weight from (is,js) to (ig,jg) < S then
        j = j+1
      end if
    until (weight from (is,js) to (ig,jg) => S)
          or (j = jmax)
    if weight from (is,js) to (ig,jg) > S then
       j = j-1
    end if
    delete all horizantal edges between (i,$j_0$) and
                                                  (i,j)
    (i,$j_0$).east_weight = (i,j).west_weight
                        = weight difference
    (i,$j_0$).east_distance = (i,j).west_distance
                          = $j-j_0$
  end loop
 end loop
```

procedure: propogation

```
for all nodes (i,j) loop
  for north loop
    while {(i,j).north_weight + (i,j).north_weight
              (i,j).north_distance} < stride loop
      (i,j).north_distance:=(i,j).north_distance + 1
       (i,j).north_weight:= old_value +
              (i,j+(i,j).north_distance).north_weight
     end loop
   end loop
end loop
repeat for other directions
```

Figure 13: The Pseudocode for Modified Grid Algorithm

The **procedure propagation** assumes that the distance between successive grid points has been already computed; otherwise a more efficient strategy could be utilized.

The modified relaxation algorithm is much faster than the traditional relaxation-based technique. The number of propagations in every polygon p_i is reduced from A_i (the area of p_i) to $A_i/(w_i**2)$, where w_i is the weight of a unit distance in p_i.

The user can change the ***threshold*** parameter in straightforward algorithm, as he can also alter the ***stride*** parameter, in addition to the ***threshold*** parameter, in the modified algorithm. The ***threshold*** parameter ensures that the algorithm stops as it is with some threshold for the optimum. The ***stride*** parameter dictates the minimum weight of an edge thus making bigger strides in low cost areas. The ***stride*** parameter indirectly controls the number of edges and nodes to be eliminated, hence making the algorithm faster.

An adventageous side-effect is the elimination of some nodes. These nodes are eliminated if they are not connected to more than one node (both source and goal points cannot be eliminated). The psuedocode for these procedures are given in the same order as in the program: Horizantal search, edge eliminating, vertical search, edge eliminating and node eliminating in Figure 14. Furthermore, a continuation to the example in Figure 12 is given in Figure 15.

```
Algorithm: To find the Weighted Region Least Cost Path
           in the modified version of relaxation-based
           grid algorithm

Given: Two dimensional grid of points: GRID
       Source point: (is,js)
       Goal point: (ig,jg)
       Threshold: T
       Stride: S

Output: Cost of minimization from source to goal: Cost
        The minimum path from source to goal:
        (is,js).....(ig,jg)

procedure:

  --This code is inserted at point xx in Figure 11
  Horizantal Search and Elimination(HSE)
  Vertical Search and Elimination(similar to HSE)
  Eliminate the nodes which are not connected to more
    than one neighbor
  --The rest of the algorithm is the same except
  --while loop uses procedure propagation to propagate
```

Figure 14: The Pseudocode for Modified Grid Algorithm

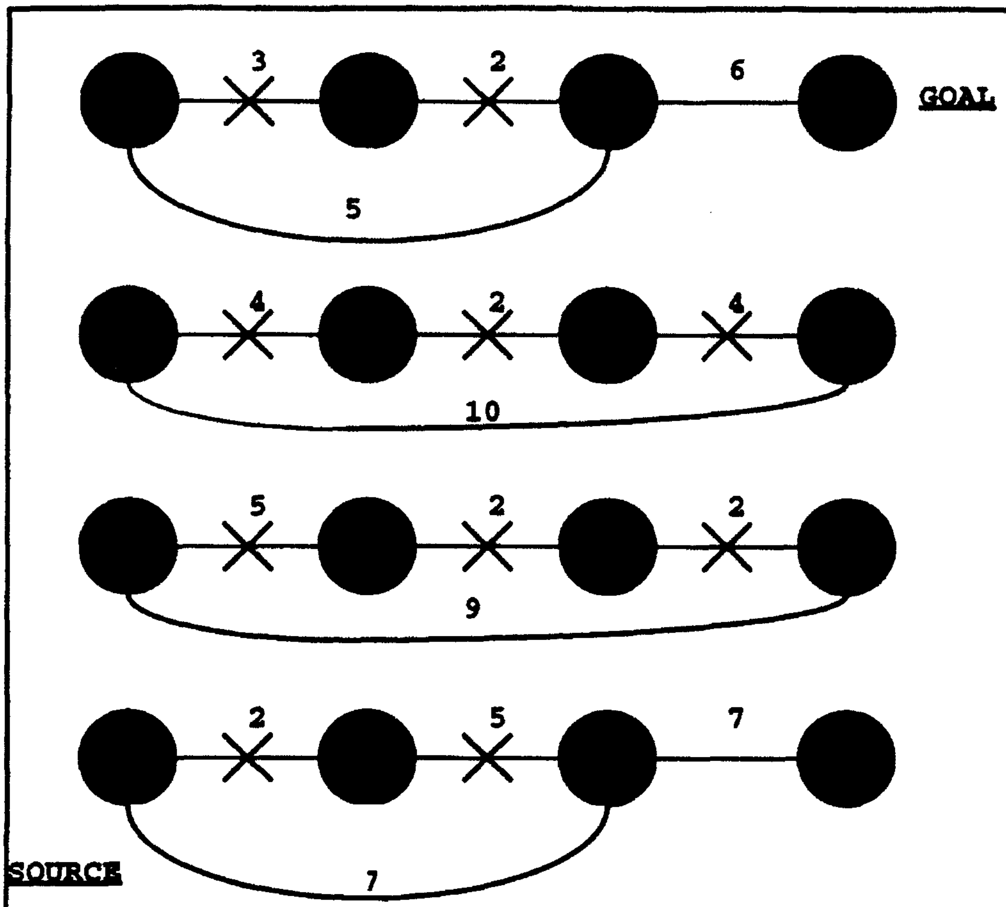
C. RESULTS

We performed a series of experiments using the Meridian Ada compiler in Sparc stations in order to realize how the parameters effect time and cost. Some of results from these experiments are combined and shown in the Table I-IV. In Table I and II we tested threshold parameter and concluded that the greater the threshold value is, the faster the program is. Changing the threshold value does not have a significant effect on the least cost. The results taken from the experiments with modified program are presented in Table III-IV. In these experiments, we kept threshold value constant (equal to zero) and changed the stride value between 1 and 100. In the first row we realized that the cost of modifications increased the amount of time for computations. As seen form the last column, the time gets less even though the second trial took longer time. The number of the nodes increases depending on the stride value. Meanwhile, the disadvantage is that the least cost gets higher for higher stride values.

D. DRAWBACKS

The modified algorithm requires more overhead. The overhead as reflected by the results is worthwhile because the overall running time of the modified algorithm is smaller than the original algorithm for an approximately close stride value. Time overhead might be reduced by not requiring that distances between all neighboring gridpoint be precomputed. Again, this algorithm is intrinsically suboptimal. There is a possibility that the graph is decomposed into more than one connected component due to the elimination of edges and nodes. If such is the case, then source and goal points might not be reachable from one another. This problem will manifest itself by having the minimum cost of the grid not changing from the assigned initial value. If there is more than one connected component, and the source and goal points are in the same connected component, then there should be a technique that avoids computations in the other connected components.

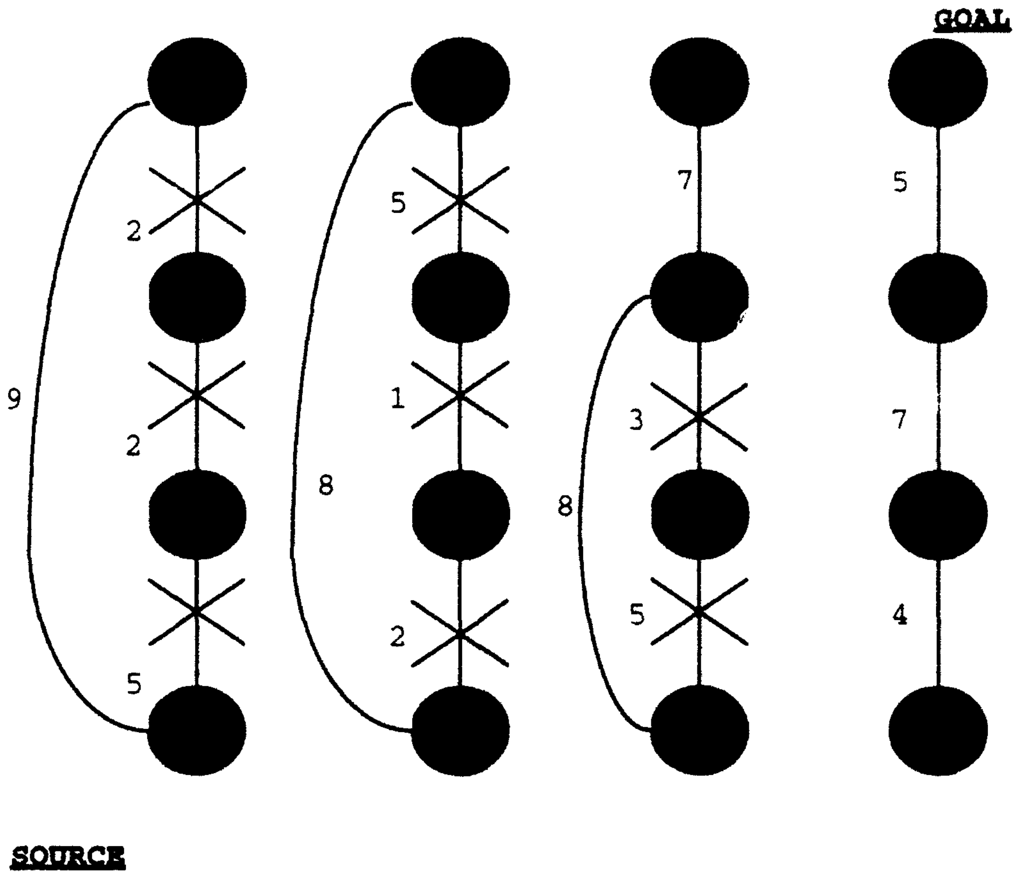
The program codes for the modified grid algorithm are enclosed in Appendix B.



STEP 1. Horizontal search is performed (e.g. with **stride** = 10, in this case the weights an edges are added until it gets more than stride.)

STEP 2. Edge Elimination is performed

Figure 15.a: An Example For A Modified Algorithm (I)

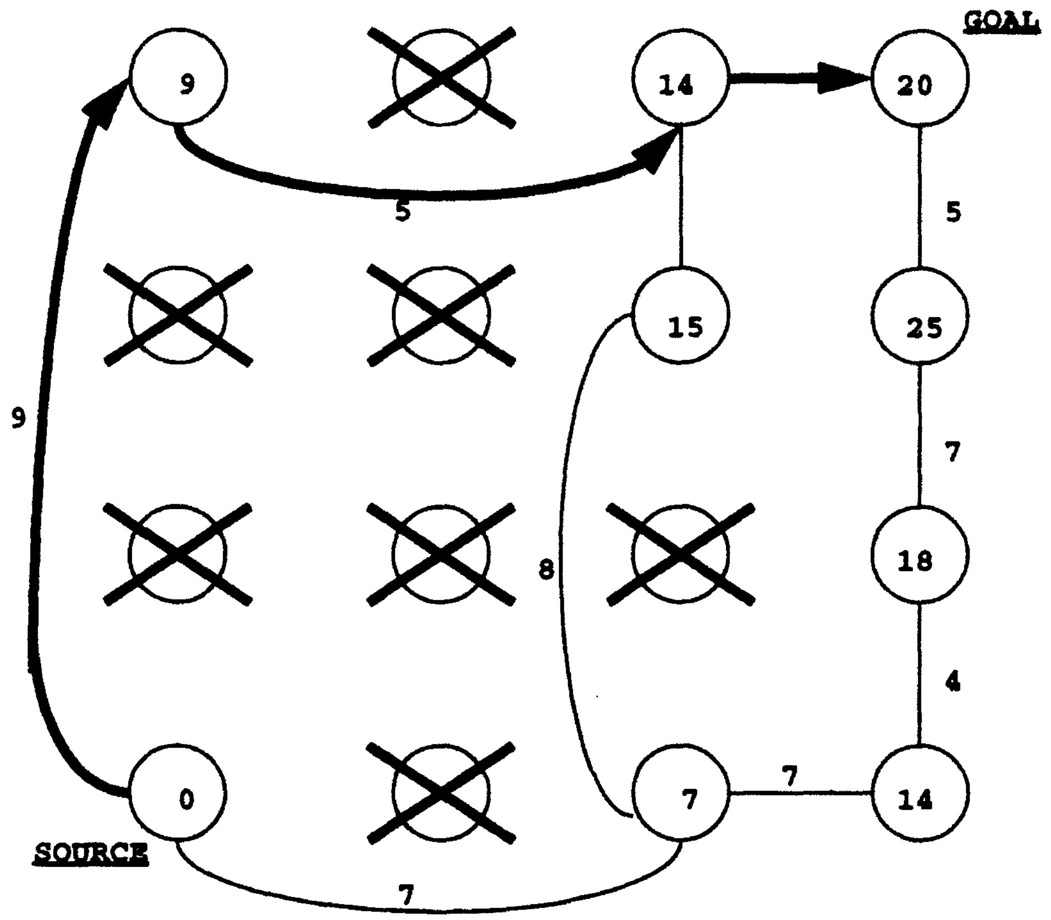


(continued from Figure 15.a)

STEP 3. Vertical Search is performed the same as horizontal search

STEP 4. Edge eliminating is performed.

Figure 15.b: An Example For A Modified Algorithm (II)



(continued from Figure 15.b)

STEP 5. Node Elimination

STEP 6. Cost Minimization and finding least cost path.

Figure 15.c: An Example For A Modified Algorithm (III)

Table 1: EXPERIMENT SHOWING THE COST AND COMPUTATION TIME VERSUS THE THRESHOLD WITH CLASSICAL ALGORITHM FROM (1,1) TO (80,80)

Threshold	Cost	Time(sec)
0	262	6.9164
10	262	5.3165
20	262	2.8832
30	262	2.8832
50	262	2.8832
100	262	2.8832

Table 2: EXPERIMENT SHOWING THE COST AND COMPUTATION TIME VERSUS THE THRESHOLD WITH CLASSICAL ALGORITHM FROM (10,1) TO (55,80)

Threshold	Cost	Time(sec)
0	308	10.0996
10	308	9.3163
20	308	9.3163
30	308	9.2996
50	308	9.2996
100	308	9.2330

Table 3: EXPERIMENT SHOWING THE COST AND COMPUTATION TIME VERSUS THE STRIDE WITH MODIFIED ALGORITHM FROM (1,1) TO (80,80)

Threshold	Stride	Cost	Time(sec)	Number of Nodes Deleted
0	1	262	11.8329	0
0	5	276	19.5826*	1375
0	10	298	10.9996	3177
0	20	330	9.3496	4747
0	30	356	7.2497	5391
0	50	356	4.0332	5877
0	100	356	2.1666	6172

Table 4: EXPERIMENT SHOWING THE COST AND COMPUTATION TIME VERSUS THE STRIDE WITH MODIFIED ALGORITHM FROM (10,1) TO (55,80)

Threshold	Stride	Cost	Time(sec)	Number of Nodes Deleted
0	1	308	17.3160	0
0	5	314	19.6659*	1375
0	10	330	10.9662	3197
0	20	336	9.3996	4730
0	30	416	7.797	5390
0	50	454	4.0498	5870
0	100	454	2.1499	6168

* : Overhead causes this increase in time

IV. THE ENVIRONMENT

In this chapter we describe both the hardware and the software environments in which we implemented the parallel algorithm presented in Chapter V. The first section introduces the Transputer [TRA87], and the second section introduces Alsys Ada [ALS90]. A guide for program development using Ada on the Transputer is presented in the third section.

A. TRANSPUTER

The Transputer implementation is based on the concept of the Communicating Sequential Processes(CSP). In order to utilize the transputer effectively, we need to understand the way it works. Transputer is a microprocessor with its own local memory storage and four links designed to communicate directly with other Transputers. The larger the number of processors in the network is, the more processing power, the more memory and links are available. The difficulties also grow with the number of processors. The most visible difficulty in the network is to avoid deadlock in which communication fails and results in processes waiting forever.

There are different types of Transputer: T2 (T212, T222), T4 (T414, T425) AND T8 (T800, T801, T805). We worked with T800 Transputers. A block diagram of a T800 Transputer is presented in Figure 15. The major components of T800 Transputer, as seen, are memory, processor and communication system connected via a 32 bit bus.

The high level programming language OCCAM [OCC88][OCC89] is the primary language used for programming the Transputers. It is designed to run concurrent processes on a network of Transputers. Concurrence and communication are two main concepts in OCCAM. They allow processes to run simultaneously and transfer information through channels from process to process. Processes communicate by message passing, do not share variables, and synchronize only when they communicate. Communication is synchronous and unbuffered.

The host computer for the Transputer network that we use is a PC-286.

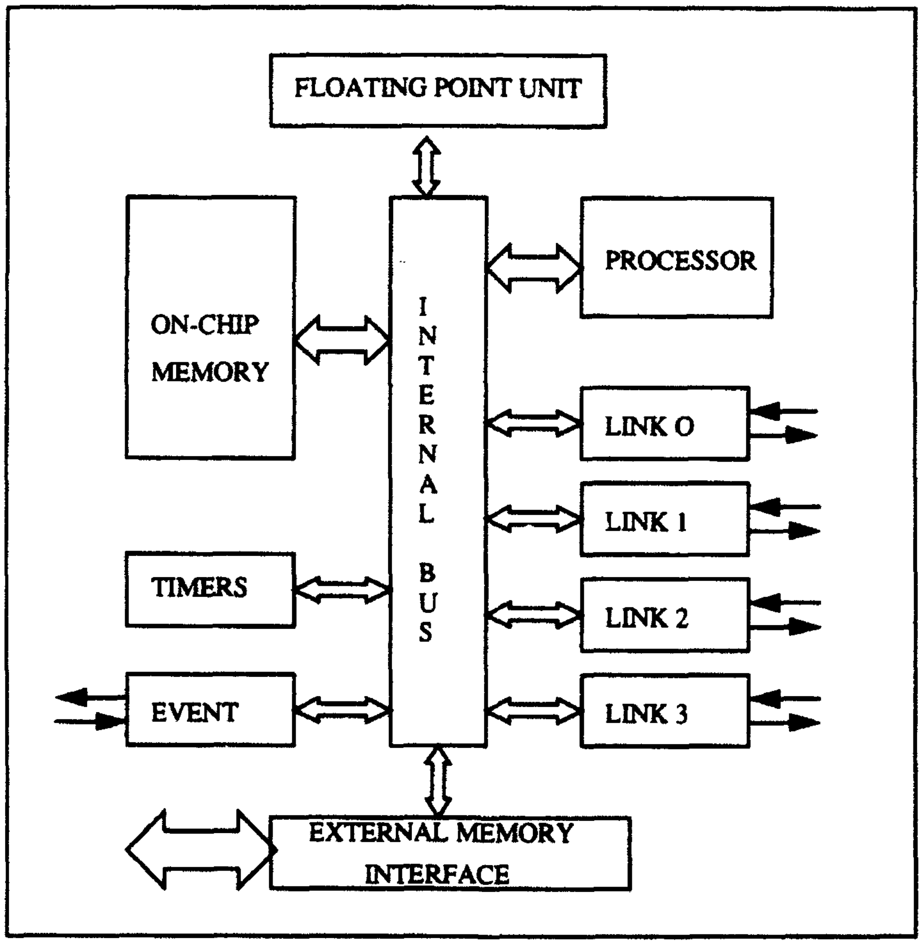


Figure 16: A Block Diagram of T800.

B. ALSYS ADA

Alsys Ada Compilation System, which consists of a compiler and a binder, is used for handling Ada programs in a Transputer programming environment. We used Alsys Ada as the language of choice for developing our WRLCP problem code. Below we describe the most salient feature that distinguishes Alsys Ada usage from ordinary programming in Ada.

1. Channels

Communication between Ada programs is provided by using transputer channels via the implementation defined package CHANNELS. The CHANNELS package contains a generic package CHANNEL_IO which defines input-output for values of a specified object type. READ, WRITE, READ_OR_FAIL, WRITE_OR_FAIL procedures within this generic package are used for input and output between channels. The distributed application is written as a set of independent programs for single or multiple Transputers and communicate through channels with unique names.

We write a COMMON package, which contains declarations common to more than one Ada program and which also contains an instantiating of CHANNEL_IO to allow data to be communicated between independent programs. The data type of the channel, common to an application, is defined in a COMMON package. For example, to declare a channel of new NAME1 of type DATA_TYPE, we include the following in the COMMON package:

```
        .
        .
        .
```

```
    DATA_TYPE : (can be any type: integer, record...);
    package NAME1 is new CHANNELS.CHANNEL_IO (DATA_TYPE);
    In the programs it is used as follows:
    declaration-- Ada program segment begins here with declaration
    NAME2:CHANNELS.CHANNEL_REF:=CHANNELS.IN_PARAMETERS(vi
rtual channel number);
    --this channel is used for input from the other processors and virtual
channel ---number is given by the programmer:e.g. 1 or 2 etc.
NAME3:CHANNELS.CHANNEL_REF:=CHANNELS.OUT_PARAMETERS(virtual
channel number); --this is for output to the other processors
    begin --main program starts here
    NAME1.READ(NAME2,the same data_type);--get the input from the
channel --NAME2
    NAME1.WRITE(NAME3,the same data_type);-- put the data_type
value to --the channel NAME3.
    end
```

2. Harnesses

In order to run Ada programs in parallel on a single processor or a multi-transputer network, we need to use an interface, which is an occam process called a *harness.* A harness is used as a wrapping for the Ada program to be accepted by a Transputer. For every Ada program, two occam harnesses have to be created. Harnesses are explained in more details in section C.

C. GUIDE FOR PROGRAM DEVELOPMENT

In this section we present some helpful points to make program development easier. After learning the MAKE Program Maintenance Utility, the tools for checking the network, and studying the examples, one will be ready for program development in this environment.

1. MAKE program maintenance Utility

MAKE is a utility program designed to help control of programming environment, to automate the process by determining which parts of the program is changed since the last compilation and rebuilds them accordingly.

makefile is a script file, written by programmer and directs MAKE. You can find MAKE commands below:

make family: Creates the Ada family and library sub-directories. This is a one-time-only operation.

make: The standard command for building the executable programs after changes have been made to the source.

make run: Executes the compiled program.

make help: Displays the MAKE commands.

make -n: Displays but do not execute commands.

make check: Checks transputer topology.

make clean: Deletes all files except source files.

make *.o: Make Ada object codes.

There is a batch file named ***doit_all*** that executes the first three commands:make family,make, and make run.

2. Checking Tools

Besides ***make check*** there are some more executable files which check the network topology: worm.exe, chknet.exe.

3. Examples in Steps of Instructions

It is better to start with complete examples to get used to it.

I. Make your own directory and copy the generic installatior into it:

>copy d:\alsys037\source\generic\*.* .

Complete documentation can be found in the files ***read.me*** and ***show.me***.

II. In this environment, there is an Ada source file ***proj.ada*** containing procedure ***proj***. If you edit your own code with these names, it means you are ready to compile your code in Alsys Ada environment.

III. You can type ***doit_all*** which executes three commands:

make family,make,make run.

IV. It is time to try the examples on a single transputer and then on multiple transputers. You can refer to the appendix C and Alsys Ada User Manuals.

V. Communicating Ada processes on a single transputer needs the list of files below:

makefile: A script file written by the programmer and executes the commands according to makefile.

family.inv: This file creates the library environment (It does not change.).

proj.inv: This file directs compiling and binding.

main.occ: In order to integrate Ada with other languages a well defined interface is required. Ada programs may then be run in parallel on a single processor or distributed across a multi-transputer network, just as occam processes. This is a default occam harness provided as part of the compilation system in both source and compiled forms.

The main body of the harness consists of three processes operating in parallel:

- A multiplexor which combines the error output and the standard output of the Ada program.

- An error channel collector which collects any output from the error stream and routes it to the standard output stream of the server via the multiplexor.

- A process which sets up the input and the output channel vectors of the Ada program and then invokes it, informing the other processes upon completion.

merger.occ: This default harness is used to collect the error output from up to some number of Ada programs and send it to the standard output stream of the server (It does not change.).

projh.occ: Each Ada (here PROJ.ADA) program has its own mini harness which provides a clean interface to the program in terms of the channels used. Main harness is used to invoke each of the mini harnesses in parallel.

projh2.occ: This is the dummy harness required to allow linking of a foreign Ada program with the occam libraries.

main.lnk: Gives the file list to link.

VI. The files needed for multiple transputers are mainly the same but they should be modified according to the network and presented below:

makefile

family.inv

proj?.inv: They should be as many as the number of transputers.

mainh.occ

merger.occ

proj?h.occ: They should be as many as the number of transputers.

proj?h2.occ: They should be as many as the number of transputers.

main.pgm: This is the only file not needed for one single transputer. It describes which virtual channels are equal to which physical channels.

main.lnk

An example is provided in Appendix C containing all these files, and it does not need to be changed for similar applications. In Figure 16 all these relations are shown.

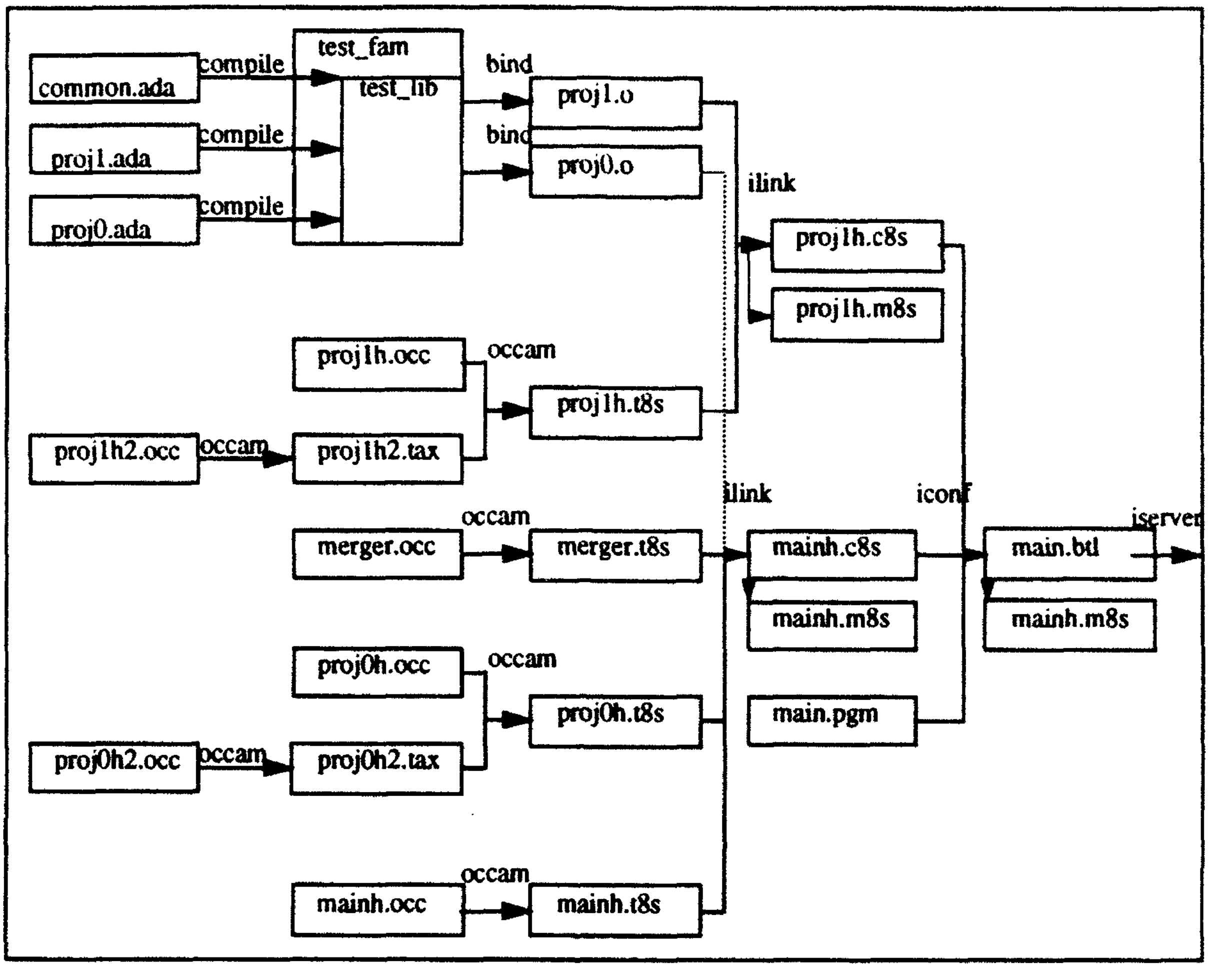


Figure 17: Diagram Of The Steps Involved In Program Development Using Alsys Ada On Transputers

V. A MULTICOMPUTER ALGORITHM

This chapter introduces a parallel algorithm for the WRLCP problem. In the first section we present the implementation on an INMOS Transputer network using Alsys Ada. In the second section we discuss a variant of that algorithm that uses less communication traffic.

A. IMPLEMENTATION

To simplify development of the algorithm, we made every possible effort to have it treat all processors symmetrically. This approach allows the algorithm to be scalable for a different number of processors without much change. We show the pseudocode for a version of the algorithm running on two processors in Figure 17.

The Root Transputer, which is the only Transputer having direct connection with the host PC, reads the data file and sends equal portions of the grid to other processors. At the beginning of the computation the Root Transputer sends the values of the threshold, the stride, the number of processors, the source and goal points to every processor. All the processors generate the weights on edges. All of them start **cost minimization** at the same time. Every processor updates the values of the grid points in its portion of the grid. For the points that are at the border of the grid portion assigned to a processor, the costs of the data obtained from the neighboring processor at the previous iteration are used. At

```
Algorithm: To find the Weighted Region Least Cost Path on
           two processors

Given: Two dimensional grid of points: GRID
       Source point: (is,js)
       Goal point: (ig,jg)
       Threshold: T

Output: Cost of minimization from source to goal: Cost
        The minimum path from source to goal:
        (is,js).....(ig,jg)

procedure:

Initialize
Read the data file
write(initial grid data)
 for all nodes loop
  calculate weight(i,j)
 end loop
write(source,goal,n,p,threshold)
flag = true
while flag loop
   flag = false
   for all nodes loop
     write(border costs) --exchanging the costs on border
     read (border costs) --nodes
     find minimum value
     change = node(i,j).old_cost-node(i,j).current_cost
     if change > T then
         raise flag
      end if
    end loop
    write(flag) --checking termination code
    read(flag)
 end loop
 read(minimized costs) --for integration the solution, it
                       --gets what the other processor did
output the WRLCP
```

Figure 18: Psuedocode For Parallel Algorithm From The One Processor's Point Of View. The Other Processor Will Have **Read** and **Write**s Interchanged.

the end of the computation of an iteration, neighboring processors exchange values of the border grid points to be used in a later iteration. After every iteration, the processors, as a whole check each other to decide whether to stop or to continue. If they decide that an adequate solution has been found they stop. All the processors write the obtained costs back to the Root Transputer which then displays the least cost path from the source to the goal points given.

The pattern for **"WRITE** and **READ"** between the processors is shown in Figure 18. Other correct patterns exist, but it should be emphasized that a very important issue is deadlock avoidance.

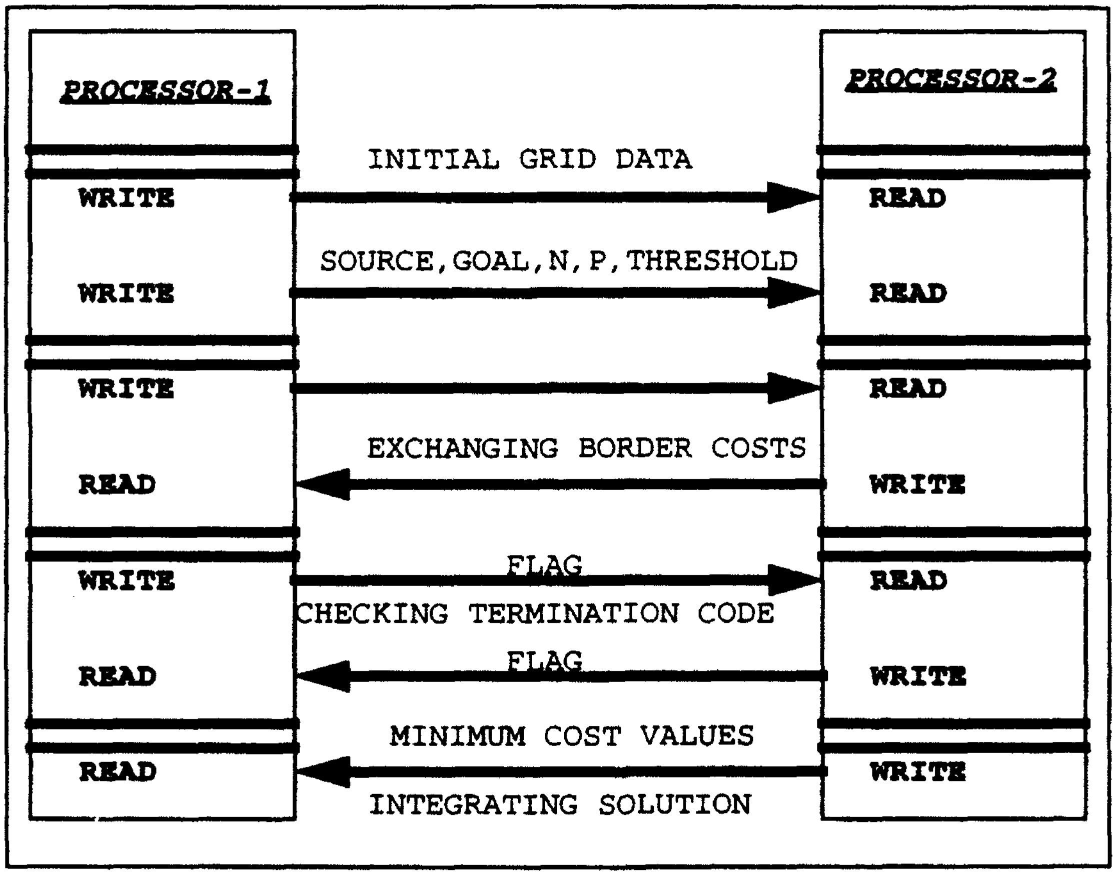


Figure 19: **Write** and **Read** Patterns

We have a running program on two processors, which is presented in appendix C. Furthermore, the methodology can be applied to more processors. Different patterns can be used to partition the grid graph on a number of processors.In Figure 19 it is shown how the grid can be partitioned in different possible patterns, e.g. for four processors.

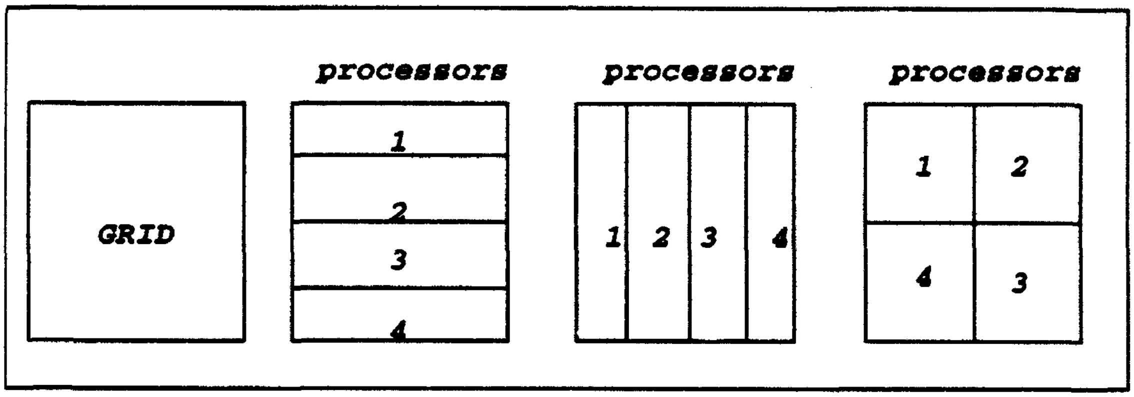


Figure 20: Patterns for Partitioning

B. A VARIANT ALGORITHM THAT SAVES ON COMMUNICATION

The cost of communication can be very large since we have to exchange all the border values and checking parameters between the neighbor processors throughout every iteration of **cost minimization**. In reality, a lot of values exchanged across the border of the grid portions are redundant because they can remain unchanged through more than one iteration. That is why saving on communication becomes very important. As we try to speed up the algorithm, the cost of communication should not slow it down, especially if part of the communication is pragmatically useless. We present a modified algorithm that takes advantage of the previous observation. In the modified approach, these border points which we changed in an iteration are marked. At the beginning of the data exchange between neighboring processors, each processor informs its neighbor about the number of the border points that changed. Then it proceeds to send the low cost values for only these points. More saving on communication can be achieved by using variant records. Unfortunately, we could not set this to work in the current development environment.The pseudocode for this approach is presented in Figure 20.

```
Algorithm: The variant algorithm that saves on
           communication

procedure:

for all border points node(i,j) loop
  change = node(i,j).old_cost - node(i,j).current_cost
  if change /= 0 then
    mark node(i,j)
  end if
end loop
count = number of marked nodes(i,j)
write(count)
for 1..count loop
   write marked node(i,j)
end loop
read(count)
for 1..count loop
   read marked node(i,j)
end loop
```

Figure 21: The Pseudocode For The Communication-Saver Variant Algorithm Of The Parallel Algorithm From The One Processor's Point Of View. The Other Processor Will Have **Read** and **Write**s Interchanged.

VI.CONCLUSIONS AND RECOMENDATIONS FOR FUTURE RESEARCH

We have developed an efficient version of a grid based algorithm to solve the WRLCP problem. Our algorithm shows a significant decrease in the computation time in comparison to the original algorithm. The experienced loss in solution accuracy is not proportional to the saving in computation time.

As a step towards developing a parallel version of this algorithm on a network of Transputers using Alsys Ada, we started developing simple parallel grid algorithm for the WRLCP problem in the above environment. Due to difficulties in dealing with that environment and due to lack of time, we stopped at the stage of developing adequate parallel algorithms for that environment, hoping that others will pursue our efforts towards the initial goal.

The main emphasis in our parallel algorithm was compile time partitioning and mapping of data. Garcia [GAR89] implemented the parallel algorithm (Local, Asycnhronous and Iterative Parallel Procedures (LAIPP) Algorithm) presented in [SMI88] on a network of Transputers using Logical C. Scheduling was dynamic by farming out computations to available processors. Garcia's results showed that at a certain point, increasing the number of processors decreased the speedup. We attribute this to excessive communication delays involved in the scheduling. We directed our effects towards nearest-neighbor patterns of communication, and we believe that this appropriate approach to handle this problem.

Many problems need yet to be solved. As a starting point, software tools for automatic generation of Ada harnesses, and automatic mapping of Ada programs need to be acquired. This, and upgrade in the existing hardware setup will bring about more rapid program development. Currently, the process of program developing in that environment is extremely tedious.

First order improvement to the existing parallel algorithms can be attained by using variant records for communication across channels.

More serious improvement include:

- designing an asynchronous version of the parallel algorithm (data will be communicated only when needed),

- using queues or heaps can decrease the computations,

- using the routing library developed in [FAL92] might provide us a way to compare our algorithm to more efficient algorithms that are not constrained to nearest neighbors.

During the course of our work, we encountered problem with theoretical flavor which yet to be solved:

- ensure that edge and node elimination in the algorithm (to reach a parallel analog of the modified sequential algorithm) does not separate the grid graph into different connected components.

LIST OF REFERENCES

[AHO83] Aho, A. V., Hopcraft, J. E., Ullman, J.D., *"Data Structures and Algorithms"*, Addison-Wesley, Inc., 1983.

[ALS90] Alsys Inc. *"PC Mothered Transputer Cross Compilation User Manuals"*, Alsys, Burlington, MA, May 1990.

[FAL92] Falcao, A. G. M., *"Allocation of Periodic Tasks With Precedences on Transputer-Based Systems"*, Master's Thesis, Naval Postgraduate School, Department of Electrical Engineering, Monterey, California, September 1992.

[GAR89] Garcia, I., *"Solving The Weighted Region Least Cost Path Problem Using Transputers"*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, December 1989.

[HAB87] Haberman, R., *"Elementary Applied Partial Differential Equations with Fourier Series and Boundary Values"*, Prentice-Hall, Inc., 1987.

[HEC87] Hecht, E., *"Optics"*, Addison-Wesley, Inc., 1987.

[KIN91] Kindl, R. M., Shing, M., Rowe, N. C., *"A Stochastic Approach To The Weighted - Region Problem: I. The Design Of The Path Annealing Algorithm II. Performance Enhancement Techniques and Experimental Results"*, Technical Report, Naval Postgraduate School, Department of Computer Science, Monterey, California, June 1991.

[MAN89] Manber,U., *"Introduction To Algorithms"*, Addison-Wesley, Inc., 1989.

[OCC88] INMOS Limited, *"Occam 2 Reference Manual"*, Prentice Hall International Ltd., 1988.

[OCC89] INMOS Limited, *"Occam 2 Toolset User Manual"*, INMOS Document Number 72 TDS 184 00, 1989.

[QUI87] Quinn, J. M., *"Designing Efficient Algorithms For Parallel Computers"*, McGraw-Hill, Inc., 1987.

[RIC87] Richbourg, F. R., *"Solving a Class of Spatial Reasoning Problems: Minimal-Cost Path Planning in the Cartesian Plane"*, Ph.D. Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, June 1987.

[SMI88] Smith, T. R., Peng, G., Gahinet, P., *"A Family of Local, Asynchronous, Iterative and Parallel*

Procedures For Solving The Weighted Region Least Cost Path Problem", Technical Report, Department of Computer Science, University of California at Santa Barbara, 20 April 1988.

[TRA87] INMOS Limited, *"Transputer Reference Manual"*, Prentice Hall Int. Ltd, January 1987.

[YUK91] Yuktadatta, P., *"Simulation Of A Parallel Processor Based Small Tactical System"*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, December 1991.

APPENDIX A: SEQUENTIAL ALGORITHM SOURCE CODE

```
--Title    : STRAIGHT-FORWARD ALGORITHM
--Author   : CENGIZ EKIN
--Date     : 20/06/92
--Revised : 04/10/92
--Course   : THESIS
--Compiler: MERIDIAN ADA
--Description:Reads data from file,input starting and goal
points   --finds the minimum cost path.
with  TEXT_IO,OS_TYPES, TASK_CONTROL, CALENDAR;
use   TEXT_IO,OS_TYPES, CALENDAR;

procedure MAIN1  is

   package INTEGER_INOUT is new INTEGER_IO(INTEGER);
   package FLOAT_INOUT is new FLOAT_IO(FLOAT);
   use  FLOAT_INOUT,INTEGER_INOUT;

   START_TIME ,
   END_TIME           : FLOAT;

   SX,SY,GX,GY,I,J       :INTEGER;
   Q                     :STRING(1..1) := "y";
   VOLTA,VOLT            :INTEGER;
   NOP                   :INTEGER :=80;
   E,COUNTER             :INTEGER;
   E1,SQ                 :STRING(1..5);
   type ELER is
       record
        WEIGHT : INTEGER;
        DISTANCE : INTEGER;
       end record;
   type GRID_POINT  is
      record
        CURRENT_COST,
        OLD_COST     : INTEGER;
        N,
        E,
        S,
        W             : ELER;
        DIRECTION     : STRING(1..5);
      end record;
   type GRID is array (0..(NOP +1),0..(NOP+1)) of GRID_POINT;
   B      : GRID;
   INF    : FILE_TYPE;

 type STORAGE is array (1..1000) of INTEGER;
 type STORAGE1 is array (1..1000) of STRING(1..5);
 type STACK is
   record
     STORE         :STORAGE;
     LATEST        :INTEGER := 0;
  end record;
type STACK1 is
   record
     STORE         :STORAGE1;
     LATEST        :INTEGER := 0;
  end record;

  S    : STACK;S1   : STACK1;
------------------------------------------------------------
--This part is for the insertion of values into the STACK.
------------------------------------------------------------
procedure PUSH (S : in out STACK; E : in INTEGER) is

 begin

    S.LATEST := S.LATEST + 1;
    S.STORE(S.LATEST) := E;
 end PUSH;
-----------------------------------------------------------
procedure PUSH1 (S1 : in out STACK1; E1 : in STRING) is

 begin

    S1.LATEST := S1.LATEST + 1;
    S1.STORE(S1.LATEST) := E1;
 end PUSH1;
------------------------------------------------------------
-- This part is to print the values from the STACK.
------------------------------------------------------------
```

```
procedure POP (S : in out STACK; E : out INTEGER) is

   begin

   E := S.STORE(S.LATEST);
   S.LATEST := S.LATEST - 1;
   end POP;
-------------------------------------------------------------
procedure POP1 (S1 : in out STACK1; E1 : out STRING) is

   begin

   E1 := S1.STORE(S1.LATEST);
   S1.LATEST := S1.LATEST - 1;
   end POP1;
-------------------------------------------------------------
  -- This function computes the execution time (CPU TIME).
-------------------------------------------------------------
    function CLOK return FLOAT is
      function CLOCK return int ;
      pragma INTERFACE(C, CLOCK);
      T   : int;
      S   : FLOAT;
      begin
        task_control.pre_emption_off;
      T := CLOCK;
      if T = -1 then
        raise TIME_ERROR;
      else
        S := FLOAT(T)/1.0E6;
      end if;
      task_control.pre_emption_on;
      return S;
    end CLOK;
-------------------------------------------------------------
procedure CAL_WEIGHT (I,J : in INTEGER) is
begin
If I = NOP   then
          B(I,J).N.WEIGHT := -1;
          B(I,J).N.DISTANCE := -1;
          B(I+1,J).CURRENT_COST :=10000;
       else
         B(I,J).N.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST -
B(I+1,J).CURRENT_COST);
       end if;
       If J = NOP then
          B(I,J).E.WEIGHT := -1;
          B(I,J).E.DISTANCE := -1;
          B(I,J+1).CURRENT_COST :=10000;
       else
         B(I,J).E.WEIGHT := 1+ABS(B(I,J).CURRENT_COST -
B(I,J+1).CURRENT_COST);
       end if;
       If I = 1 then
          B(I,J).S.WEIGHT := -1;
          B(I,J).S.DISTANCE := -1;
          B(I-1,J).CURRENT_COST :=10000;
       else
         B(I,J).S.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST - B(I-
1,J).CURRENT_COST);
       end if;
       If J = 1 then
          B(I,J).W.WEIGHT := -1;
          B(I,J).W.DISTANCE := -1;
          B(I,J-1).CURRENT_COST :=10000;
       else
        B(I,J).W.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST - B(I,J-
1).CURRENT_COST);
       end if;
 end CAL_WEIGHT;
------------------------------------------------------------
procedure FIND_MIN (I,J : in INTEGER) is
begin
if B(I,J).CURRENT_COST >  (B(I+1,J).CURRENT_COST
                              +abs(B(I,J).N.WEIGHT)) then
          B(I,J).CURRENT_COST := (B(I+1,J).CURRENT_COST
                               +abs(B(I,J).N.WEIGHT));
          B(I,J).DIRECTION    := "NORTH";
       end if;
       if B(I,J).CURRENT_COST >  (B(I,J+1).CURRENT_COST
                              +abs(B(I,J).E.WEIGHT)) then
          B(I,J).CURRENT_COST := (B(I,J+1).CURRENT_COST
                               +abs(B(I,J).E.WEIGHT));
          B(I,J).DIRECTION    := "EAST ";
       end if;
       if B(I,J).CURRENT_COST >  (B(I-1,J).CURRENT_COST
                                 +abs(B(I,J).S.WEIGHT)) then
         B(I,J).CURRENT_COST := (B(I-1,J).CURRENT_COST
                                 +abs(B(I,J).S.WEIGHT));
         B(I,J).DIRECTION    := "SOUTH";
      end if;
      if B(I,J).CURRENT_COST >  (B(I,J-1).CURRENT_COST
                                 +abs(B(I,J).W.WEIGHT)) then
         B(I,J).CURRENT_COST := (B(I,J-1).CURRENT_COST
                                 +abs(B(I,J).W.WEIGHT));
         B(I,J).DIRECTION    := "WEST ";
      end if;
end FIND_MIN;
------------------------------------------------------------
-- Main program ...
------------------------------------------------------------
begin

  while Q = "y" loop
    COUNTER := 1;
    for I in  1 .. NOP loop
      for J in  1 .. NOP  loop
         B(I,J).N.DISTANCE :=1;
         B(I,J).E.DISTANCE :=1;
         B(I,J).S.DISTANCE :=1;
         B(I,J).W.DISTANCE :=1;
      end loop;
   end loop;
   OPEN (INF,MODE => IN_FILE, NAME => "ter.dat");
   PUT ("THE DIMENSION OF MATRIX    =" );
   GET (NOP);
   PUT_LINE ("ENTER THE VOLTA (the optimization tolerance)");
   PUT ("VOLTA = ");GET (VOLTA);
   PUT_LINE ("ENTER THE SOURCE POINT !");
   PUT("SX = ");GET (SX);PUT("SY = ");GET(SY);
   PUT_LINE ("ENTER THE GOAL POINT !");
   PUT("GX = ");GET (GX);PUT("GY = ");GET(GY);
-----------------------------------------------------------
 -- CLOK function begins to compute the execution time.
   START_TIME := CLOK;
------------------------------------------------------------
--This part gets the heights from the file..
------------------------------------------------------------
   for ROW in 1..NOP loop
   for COL in 1..NOP loop
    GET (INF, B(ROW,COL).CURRENT_COST);
   end loop;
  end loop;
------------------------------------------------------------
--CLOK function finishes the computation of execution time.
 END_TIME := CLOK;
 PUT("TOTAL TIME TO READ THE DATA FILE IS :  ");
 PUT(END_TIME-START_TIME,4,4,0);new_line;
 START_TIME := CLOK;
------------------------------------------------------------
--It determines the borders and calculates the weights of the
edges..
------------------------------------------------------------
 for I in 1..NOP loop
    for J in 1..NOP loop
        CAL_WEIGHT(I,J);
    end loop;
 end loop;
-----------------------------------------------------------
--It makes the costs max number in order to use them in
comparisons for finding
-- the minimum....
------------------------------------------------------------
  for I in  1 .. NOP loop
     for J in  1 .. NOP  loop
        B(I,J).CURRENT_COST := 10000;
        B(I,J).OLD_COST :=  B(I,J).CURRENT_COST;
     end loop;
  end loop;
------------------------------------------------------------
--cost minimization...
------------------------------------------------------------
 B(SX,SY).CURRENT_COST := 0;
 while COUNTER > 0 loop
    COUNTER := 0;
    for I in 1..NOP loop
      for J in 1..NOP loop
         FIND_MIN(I,J);
         VOLT := B(I,J).OLD_COST -B(I,J).CURRENT_COST;
         if VOLT > VOLTA then
            COUNTER := COUNTER +1;
         end if;
```

```
          B(I,J).OLD_COST := B(I,J).CURRENT_COST;
      end loop;
    end loop;
end loop;
------------------------------------------------------------
--output of least cost path..
------------------------------------------------------------
loop
   PUSH(S,B(GX,GY).CURRENT_COST);
   PUSH1  (S1,B(GX,GY).DIRECTION);
   SQ :=B(GX,GY).DIRECTION;
   if SQ = "NORTH" then
      PUSH(S,B(GX,GY).N.DISTANCE);
      GX := GX+1;
   elsif SQ = "EAST " then
      PUSH(S,B(GX,GY).E.DISTANCE);
      GY := GY+1;
   elsif SQ = "SOUTH" then
      PUSH(S,B(GX,GY).S.DISTANCE);
      GX := GX-1;
   elsif SQ =  "WEST " then
      PUSH(S,B(GX,GY).W.DISTANCE);
      GY := GY-1;
   else
   exit;
   end if;
exit when GX =SX and GY =SY ;
end loop;
------------------------------------------------------------
PUT_LINE("       DISTANCE      COST   DIRECTION");
PUT_LINE("       --------      ----   ----------");
loop
  POP(S,E);PUT(E);PUT ("   ");
  POP(S,E);PUT(E);PUT ("      ");
  POP1(S1,E1);
  if E1 = "NORTH" then
      PUT ("SOUTH");new_line;
   elsif E1 = "EAST " then
      PUT ("WEST");new_line;
   elsif E1 = "SOUTH" then
      PUT ("NORTH");new_line;
   elsif E1 =  "WEST " then
      PUT ("EAST");new_line;
   end if;

  exit when S.LATEST =0;
end loop;
------------------------------------------------------------
CLOSE(INF);
--CLOK function finishes the computation of execution time.
END_TIME := CLOK;
PUT("TOTAL TIME TO EXECUTE THE PROGRAM IS :  ");
PUT(END_TIME-START_TIME,4,4,0);new_line;
PUT("RUN ONE MORE TIME :y(es) or n(o) :");GET(Q);
if Q = "n" then exit;
end if;
end loop;
end MAIN1;
```

APPENDIX B: MODIFIED ALGORITHM SOURCE CODE

```
--Title    : MODIFIED ALGORITHM
--Author   : CENGIZ EKIN
--Date     : 20/06/92
--Revised  : 04/10/92
--Course   : THESIS
--Compiler: MERIDIAN ADA
--Description:modified program, searches and deletes nodes
and ----edges
with  TEXT_IO,OS_TYPES, TASK_CONTROL, CALENDAR;
use   TEXT_IO,OS_TYPES, CALENDAR;

 procedure MAIN2  is

   package INTEGER_INOUT is new INTEGER_IO(INTEGER);
   package FLOAT_INOUT is new FLOAT_IO(FLOAT);
   use  FLOAT_INOUT,INTEGER_INOUT;

   START_TIME ,
   END_TIME         : FLOAT;


   SX,SY,GX,GY,Z,CO2,CO3,CO4,CO5 :INTEGER;
   E,COUNTER,I,J,L               :INTEGER;
   E1,SQ                         :STRING(1..5);
   Q                             :STRING(1..1) := "y";
   DEL1,A,VOLTA,VOLT             :INTEGER;
   NOP                           :INTEGER := 80;--500;
   MARK                          :BOOLEAN;
   type ELER is
       record
        WEIGHT : INTEGER;
        DISTANCE : INTEGER;
       end record;
   type GRID_POINT  is
      record
        CURRENT_COST,
        OLD_COST    : INTEGER;
        N,
        E,
        S,
        W            : ELER;
        DIRECTION     : STRING(1..5);
        ACTIVE        : BOOLEAN;
      end record;
   type GRID is array (0..(NOP +1),0..(NOP+1)) of GRID_POINT;
   B      :GRID;
   INF    : FILE_TYPE;

 type STORAGE is array (1..1000) of INTEGER;
 type STORAGE1 is array (1..1000) of STRING(1..5);
 type STACK is
   record
     STORE           :STORAGE;
     LATEST          :INTEGER := 0;
  end record;
type STACK1 is
   record
     STORE           :STORAGE1;
     LATEST          :INTEGER := 0;
  end record;

  S    : STACK;S1   : STACK1;
-----------------------------------------------------------
--This part is for the insertion of values into the STACK.
-----------------------------------------------------------
procedure PUSH (S : in out STACK; E : in INTEGER) is

 begin

    S.LATEST := S.LATEST + 1;
    S.STORE(S.LATEST) := E;
 end PUSH;
-----------------------------------------------------------
procedure PUSH1 (S1 : in out STACK1; E1 : in STRING) is

 begin

    S1.LATEST := S1.LATEST + 1;
    S1.STORE(S1.LATEST) := E1;
 end PUSH1;
```

```
-----------------------------------------------------------
-- This part is to print the values from the STACK.
-----------------------------------------------------------
procedure POP (S : in out STACK; E : out INTEGER) is

   begin

   E := S.STORE(S.LATEST);
   S.LATEST := S.LATEST - 1;
   end POP;
-----------------------------------------------------------
procedure POP1 (S1 : in out STACK1; E1 : out STRING) is

   begin

   E1 := S1.STORE(S1.LATEST);
   S1.LATEST := S1.LATEST - 1;
   end POP1;
-----------------------------------------------------------
  -- This function computes the execution time (CPU TIME).
-----------------------------------------------------------
    function CLOK return FLOAT is
      function CLOCK return int ;
      pragma INTERFACE(C, CLOCK);
      T  : int;
      S  : FLOAT;

      begin
        task_control.pre_emption_off;

      T := CLOCK;

      if T = -1 then
        raise TIME_ERROR;
      else
        S := FLOAT(T)/1.0E6;
      end if;

      task_control.pre_emption_on;
      return S;
    end CLOK;
-----------------------------------------------------------
procedure CAL_WEIGHT (I,J : in INTEGER) is
```

```
begin
If I = NOP   then
          B(I,J).N.WEIGHT := -1;
          B(I,J).N.DISTANCE := -1;
          B(I+1,J).CURRENT_COST :=10000;
       else
         B(I,J).N.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST -
B(I+1,J).CURRENT_COST);
       end if;
       If J = NOP then
          B(I,J).E.WEIGHT := -1;
          B(I,J).E.DISTANCE := -1;
          B(I,J+1).CURRENT_COST :=10000;
       else
         B(I,J).E.WEIGHT := 1+ABS(B(I,J).CURRENT_COST -
B(I,J+1).CURRENT_COST);
       end if;
       If I = 1 then
          B(I,J).S.WEIGHT := -1;
          B(I,J).S.DISTANCE := -1;
          B(I-1,J).CURRENT_COST :=10000;
       else
         B(I,J).S.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST - B(I-
1,J).CURRENT_COST);
       end if;
       If J = 1 then
          B(I,J).W.WEIGHT := -1;
          B(I,J).W.DISTANCE := -1;
          B(I,J-1).CURRENT_COST :=10000;
       else
        B(I,J).W.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST - B(I,J-
1).CURRENT_COST);
       end if;
 end CAL_WEIGHT;
------------------------------------------------------------
procedure FIND_MIN (I,J : in INTEGER) is
begin
       if B(I,J).ACTIVE = TRUE then
       if B(I,J).N.DISTANCE /= -1 then
       if B(I,J).CURRENT_COST >
(B((I+abs(B(I,J).N.DISTANCE)),J).CURRENT_COST
                                 +abs(B(I,J).N.WEIGHT)) then
          B(I,J).CURRENT_COST :=
(B((I+abs(B(I,J).N.DISTANCE)),J).CURRENT_COST
                                  +abs(B(I,J).N.WEIGHT));
          B(I,J).DIRECTION    := "NORTH";
       end if;
       end if;
       if B(I,J).E.DISTANCE /= -1 then
       if B(I,J).CURRENT_COST >
(B(I,(J+abs(B(I,J).E.DISTANCE))).CURRENT_COST
                                 +abs(B(I,J).E.WEIGHT)) then
          B(I,J).CURRENT_COST :=
(B(I,(J+abs(B(I,J).E.DISTANCE))).CURRENT_COST
                                  +abs(B(I,J).E.WEIGHT));
          B(I,J).DIRECTION    := "EAST ";
       end if;
       end if;
       if B(I,J).S.DISTANCE /= -1 then
       if B(I,J).CURRENT_COST >  (B((I-
abs(B(I,J).S.DISTANCE)),J).CURRENT_COST
                                 +abs(B(I,J).S.WEIGHT)) then
         B(I,J).CURRENT_COST := (B((I-
abs(B(I,J).S.DISTANCE)),J).CURRENT_COST
                                 +abs(B(I,J).S.WEIGHT));
         B(I,J).DIRECTION    := "SOUTH";
      end if;
      end if;
      if B(I,J).W.DISTANCE /= -1 then
      if B(I,J).CURRENT_COST >  (B(I,(J-
abs(B(I,J).W.DISTANCE))).CURRENT_COST
                                 +abs(B(I,J).W.WEIGHT)) then
         B(I,J).CURRENT_COST := (B(I,(J-
abs(B(I,J).W.DISTANCE))).CURRENT_COST
                                 +abs(B(I,J).W.WEIGHT));
         B(I,J).DIRECTION    := "WEST ";
      end if;
      end if;
      end if;
end FIND_MIN;
------------------------------------------------------------
-- Mair. program ...
------------------------------------------------------------
begin
  while Q = "y" loop
   MARK := TRUE;
   COUNTER := 1;
   for I in  1 .. NOP loop
      for J in  1 .. NOP  loop
         B(I,J).N.DISTANCE :=1;
         B(I,J).E.DISTANCE :=1;
         B(I,J).S.DISTANCE :=1;
         B(I,J).W.DIST\NCE :=1;
         B(I,J).ACTIVE      :=TRUE;
      end loop;
   end loop;
   OPEN (INF,MODE => IN_FILE, NAME => "ter.dat");
   PUT ("THE DIMENSION OF MATRIX    =" );
   GET (NOP);
   PUT_LINE ("ENTER THE DELTA (the jumping value).");
   PUT ("DEL1 = ");GET (DEL1);
  PUT_LINE ("ENTER THE VOLTA (the optimization tolerance).");
   PUT ("VOLTA = ");GET (VOLTA);
   PUT_LINE ("ENTER THE SOURCE POINT !");
   PUT("SX = ");GET (SX);PUT("SY = ");GET(SY);
   PUT_LINE ("ENTER THE GOAL POINT !");
   PUT("GX = ");GET (GX);PUT("GY = ");GET(GY);

 -- CLOK function begins to compute the execution time.
   START_TIME := CLOK;

-----------------------------------------------------------
--This part gets the heights from the file..
-----------------------------------------------------------
   for ROW in 1..NOP loop
     for COL in 1..NOP loop
       GET (INF, B(ROW,COL).CURRENT_COST);
     end loop;
   end loop;
-----------------------------------------------------------
--CLOK function finishes the computation of execution time.
  END_TIME := CLOK;

  PUT("TOTAL TIME TO READ THE DATA FILE IS :  ");
  PUT(END_TIME-START_TIME,4,4,0);new_line;
  START_TIME := CLOK;
-----------------------------------------------------------
--It determines the borders and calculates the weights of the
edges..
```

```
------------------------------------------------------------
 for I in 1..NOP loop
     for J in 1..NOP loop
        CAL_WEIGHT(I,J);
     end loop;
  end loop;
------------------------------------------------------------
--It makes the costs max number in order to use them in
comparisons              --for finding the minimum....
------------------------------------------------------------
   for I in  1 .. NOP loop
      for J in  1 .. NOP  loop
         B(I,J).CURRENT_COST := 10000;
         B(I,J).OLD_COST :=  B(I,J).CURRENT_COST;
      end loop;
   end loop;
------------------------------------------------------------
--horizontal search..
------------------------------------------------------------
B(SX,SY).CURRENT_COST := 0; I:=1;J:=1;
loop
  A    := B(I,J).E.WEIGHT;
  CO2  := 0;
  if A <=  DEL1  then
     loop
       exit when (A > DEL1) ;
       if (J>=NOP) then exit; end if;
      if((I=SX and J=SY) or (I=GX and J=GY)) and (MARK =TRUE)
then
         MARK := FALSE;exit; end if;
       MARK := TRUE;
       J   := J + 1;
       CO2 := CO2 + 1;
       A   := A + B(I,J).E.WEIGHT;
     end loop;
     if CO2 > 0 then
     B(I,J-CO2).E.WEIGHT   := A - B(I,J).E.WEIGHT;
     B(I,J-CO2).E.DISTANCE := CO2;
     B(I,J).W.WEIGHT       := B(I,J-CO2).E.WEIGHT;
B(I,J).W.DISTANCE     := B(I,J-CO2).E.DISTANCE;
     end if;
  else
     J := J + 1;
  end if;
  if J = NOP then
     I := I + 1;
     J := 1;
  end if;
  exit when I = NOP + 1;
end loop;
-------------------------------------------------------------
-- horizontal edge eliminating...
-------------------------------------------------------------
CO3 := 0; J := 1;I :=1;
loop
    if B(I,J).E.DISTANCE > 1 then
       Z := B(I,J).E.DISTANCE + J;
       loop
          J := J+1;
          exit when  J = Z;
          B(I,J).E.DISTANCE := -1;
          B(I,J).W.DISTANCE := -1;
          CO3 := CO3 + 2;
        end loop;
     else
     J := J+1;
     end if;
     if J = NOP then
     I := I + 1;
     J := 1;
     end if;
     exit when I = NOP + 1;
end loop;
-------------------------------------------------------------
--vertical search...
-------------------------------------------------------------
I := 1;J:=1;MARK := TRUE;
loop
  A    := B(I,J).N.WEIGHT;
  CO2  := 0;
  if A <=  DEL1 then
     loop
       exit when (A > DEL1) ;
       if (I>=NOP) then exit; end if;
      if((I=SX and J=SY) or (I=GX and J=GY)) and (MARK =TRUE)
then
```

```
         MARK := FALSE;exit; end if;
       MARK := TRUE;
       I   := I + 1;
       CO2 := CO2 + 1;
       A   := A + B(I,J).N.WEIGHT;
     end loop;
     if CO2 > 0 then
     B(I-CO2,J).N.WEIGHT   := A - B(I,J).N.WEIGHT;
     B(I-CO2,J).N.DISTANCE := CO2;
     B(I,J).S.WEIGHT       := B(I-CO2,J).N.WEIGHT;
     B(I,J).S.DISTANCE     := B(I-CO2,J).N.DISTANCE;
     end if;
  else
     I   := I + 1;
  end if;
  if I = NOP then
     J := J + 1;
     I := 1;
  end if;
  exit when J = NOP + 1;
end loop;
------------------------------------------------------------
--vertical edge eliminating...
------------------------------------------------------------
I := 1;J:=1;
loop
    if B(I,J).N.DISTANCE > 1 then
       Z := B(I,J).N.DISTANCE + I;
       loop
          I := I+1;
          exit when I = Z;
          B(I,J).N.DISTANCE := -1;
          B(I,J).S.DISTANCE := -1;
          CO3 := CO3 + 2;
       end loop;
     else
     I := I+1;
     end if;
     if I = NOP then
     J := J + 1;
     I := 1;
     end if;
     exit when J = NOP + 1;
```

```
end loop;
------------------------------------------------------------
--node eliminating ...
------------------------------------------------------------
CO5 := 0;J :=1;I:=1;
for I in 1 .. NOP loop
 if not(I=SX and I=SY) and not(I=GX and I=GY) then
      CO4 := 0;
      if B(I,I).N.DISTANCE < 1 then
         CO4 := CO4 + 1;
      end if;
      if B(I,I).E.DISTANCE < 1 then
         CO4 := CO4 + 1;
      end if;
      if B(I,I).S.DISTANCE < 1 then
         CO4 := CO4 + 1;
      end if;
      if B(I,I).W.DISTANCE < 1 then
         CO4 := CO4 + 1;
      end if;
      if CO4 >= 3 then
         B(I,I).ACTIVE := FALSE;
         if B(I,I).N.DISTANCE /= -1 then
           B((I+abs(B(I,I).N.DISTANCE)),I).S.DISTANCE := -1;
            B(I,I).N.DISTANCE := -1;
            CO3 :=CO3 + 1;
         end if;

         if B(I,I).E.DISTANCE /= -1 then
           B(I,(I+abs(B(I,I).E.DISTANCE))).W.DISTANCE := -1;
            B(I,I).E.DISTANCE := -1;
            CO3 :=CO3 + 1;
         end if;
         if B(I,I).S.DISTANCE /= -1 then
           B((I-abs(B(I,J).S.DISTANCE)),J).N.DISTANCE := -1;
            B(I,I).S.DISTANCE := -1;
            CO3 :=CO3 + 1;
         end if;
         if B(I,I).W.DISTANCE /= -1 then
          B(I,(I-abs(B(I,I).W.DISTANCE))).E.DISTANCE := -1;
            B(I,I).W.DISTANCE := -1;
            CO3 :=CO3 + 1;
         end if;
      CO5 := CO5 +1;
   end if;
 end if;
for J in I+1 .. NOP loop
   CO4 := 0;
  if not(I=SX and J=SY) and not(I=GX and J=GY) then
   if B(I,J).N.DISTANCE < 1 then
      CO4 := CO4 + 1;
   end if;
   if B(I,J).E.DISTANCE < 1 then
      CO4 := CO4 + 1;
   end if;
   if B(I,J).S.DISTANCE < 1 then
      CO4 := CO4 + 1;
   end if;
   if B(I,J).W.DISTANCE < 1 then
      CO4 := CO4 + 1;
   end if;
   if CO4 >= 3 then
      B(I,J).ACTIVE := FALSE;
      if B(I,J).N.DISTANCE /= -1 then
        B((I+abs(B(I,J).N.DISTANCE)),J).W.DISTANCE := -1;
         B(I,J).N.DISTANCE := -1;
         CO3 :=CO3 + 1;
      end if;
      if B(I,J).E.DISTANCE /= -1 then
        B(I,(J+abs(B(I,J).E.DISTANCE))).W.DISTANCE := -1;
         B(I,J).E.DISTANCE := -1;
         CO3 :=CO3 + 1;
      end if;
      if B(I,J).S.DISTANCE /= -1 then
        B((I-abs(B(I,J).S.DISTANCE)),J).N.DISTANCE := -1;
         B(I,J).S.DISTANCE := -1;
         CO3 :=CO3 + 1;
      end if;
      if B(I,J).W.DISTANCE /= -1 then
        B(I,(J-abs(B(I,J).W.DISTANCE))).E.DISTANCE := -1;
         B(I,J).W.DISTANCE := -1;
         CO3 :=CO3 + 1;
      end if;
      CO5 := CO5 +1;
   end if;
   end if;
```

```
      CO4 := 0;
     if not(J=SX and I=SY) and not(J=GX and I=GY) then
      if B(J,I).N.DISTANCE < 1 then
         CO4 := CO4 + 1;
      end if;
      if B(J,I).E.DISTANCE < 1 then
         CO4 := CO4 + 1;
      end if;
      if B(J,I).S.DISTANCE < 1 then
         CO4 := CO4 + 1;
      end if;
      if B(J,I).W.DISTANCE < 1 then
         CO4 := CO4 + 1;
      end if;
      if CO4 >= 3 then
         B(J,I).ACTIVE := FALSE;
         if B(J,I).N.DISTANCE /= -1 then
          B((J+abs(B(J,I).N.DISTANCE)),I).S.DISTANCE := -1;
            B(J,I).N.DISTANCE := -1;
            CO3 :=CO3 + 1;
         end if;
         if B(J,I).E.DISTANCE /= -1 then
           B(J,(I+abs(B(J,I).E.DISTANCE))).W.DISTANCE := -1;
            B(J,I).E.DISTANCE := -1;
            CO3 :=CO3 + 1;
         end if;
         if B(J,I).S.DISTANCE /= -1 then
           B((J-abs(B(J,I).S.DISTANCE)),I).N.DISTANCE := -1;
            B(J,I).S.DISTANCE := -1;
            CO3 :=CO3 + 1;
         end if;
         if B(J,I).W.DISTANCE /= -1 then
           B(J,(I-abs(B(J,I).W.DISTANCE))).E.DISTANCE := -1;
            B(J,I).W.DISTANCE := -1;
            CO3 :=CO3 + 1;
         end if;
         CO5 := CO5 +1;
       end if;
      end if
   end loop;
 end loop;
put(co5);
------------------------------------------------------------
```

```
--cost minimization...
------------------------------------------------------------
  while COUNTER > 0 loop
     COUNTER := 0;
     for I in 1..NOP loop
       for J in 1..NOP loop
          FIND_MIN(I,J);
          VOLT := B(I,J).OLD_COST -B(I,J).CURRENT_COST;
          if VOLT  > VOLTA then
             COUNTER := COUNTER +1;
          else
             COUNTER := COUNTER;
          end if;
          B(I,J).OLD_COST := B(I,J).CURRENT_COST;
      end loop;
    end loop;
end loop;
------------------------------------------------------------
--output of least cost path..
------------------------------------------------------------
loop
   PUSH(S,B(GX,GY).CURRENT_COST);
   PUSH1 (S1,B(GX,GY).DIRECTION);
   SQ :=B(GX,GY).DIRECTION;
   if SQ = "NORTH" then
      PUSH(S,B(GX,GY).N.DISTANCE);
      GX := GX+B(GX,GY).N.DISTANCE;
   elsif SQ = "EAST " then
      PUSH(S,B(GX,GY).E.DISTANCE);
      GY := GY+B(GX,GY).E.DISTANCE;
   elsif SQ = "SOUTH" then
      PUSH(S,B(GX,GY).S.DISTANCE);
      GX := GX-B(GX,GY).S.DISTANCE;
   elsif SQ =  "WEST " then
      PUSH(S,B(GX,GY).W.DISTANCE);
      GY := GY-B(GX,GY).W.DISTANCE;
   else
   exit;
   end if;
exit when GX =SX and GY =SY ;
end loop;
------------------------------------------------------------
PUT_LINE("       DISTANCE      COST   DIRECTION");
PUT_LINE("       --------      ----   ---------");
loop
  POP(S,E);PUT(E);PUT ("   ");
  POP(S,E);PUT(E);PUT ("      ");
  POP1(S1,E1);
  if E1 = "NORTH" then
      PUT ("SOUTH");new_line;
   elsif E1 = "EAST " then
      PUT ("WEST");new_line;
   elsif E1 = "SOUTH" then
      PUT ("NORTH");new_line;
   elsif E1 =  "WEST " then
      PUT ("EAST");new_line;
   end if;

  exit when S.LATEST =0;
end loop;
------------------------------------------------------------
CLOSE(INF);
--CLOK function finishes the computation of execution time.
END_TIME := CLOK;
PUT("TOTAL TIME TO EXECUTE THE PROGRAM IS :  ");
PUT(END_TIME-START_TIME,4,4,0);new_line;
PUT("RUN ONE MORE TIME :y(es) or n(o) :");GET(Q);
if Q = "n" then exit;
end if;
end loop;
end MAIN2;
```

APPENDIX C: PARALLEL ALGORITHM SOURCE CODE

```
--Title    : PROJ0.ADA
--Author   : CENGIZ EKIN
--Date     : 20/06/92
--Revised : 04/10/92
--Course   : THESIS
--Compiler:  ALSYS ADA
--Description:Reads data from file,input starting and goal
-- points,finds the minimum cost path.
with  TEXT_IO, COMMON, CHANNELS;
use   COMMON;
procedure PROJ0  is
   package INTEGER_INOUT is new TEXT_IO.INTEGER_IO(INTEGER);
   package FLOAT_INOUT is new TEXT_IO.FLOAT_IO(FLOAT);
   use  FLOAT_INOUT,INTEGER_INOUT;
   INF     : TEXT_IO.FILE_TYPE;
   B       : GRID;GRI :GRID_POINT;
   counter : integer :=1;
   -- communication channels that are used
   OutToMars   : CHANNELS.CHANNEL_REF :=
CHANNELS.OUT_PARAMETERS (2);
   InFromMars : CHANNELS.CHANNEL_REF :=
CHANNELS.IN_PARAMETERS (2);
------------------------------------------------------------
--This procedure calculates the weights on edges and builds
up an hyphotetical
--wall for processes to stay in the terrain...
-----------------------------------------------------------
procedure CAL_WEIGHT (I,J : in INTEGER) is
begin
      If I = 10   then
          B(I,J).N.WEIGHT := -1;
          B(I,J).N.DISTANCE := -1;
          B(I+1,J).CURRENT_COST :=10000;
       else
         B(I,J).N.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST -
B(I+1,J).CURRENT_COST);
       end if;
       If J = 10 then
```

```
          B(I,J).E.WEIGHT := -1;
          B(I,J).E.DISTANCE := -1;
          B(I,J+1).CURRENT_COST :=10000;
       else
         B(I,J).E.WEIGHT := 1+ABS(B(I,J).CURRENT_COST -
B(I,J+1).CURRENT_COST);
       end if;
       If I = 1 then
          B(I,J).S.WEIGHT := -1;
          B(I,J).S.DISTANCE := -1;
          B(I-1,J).CURRENT_COST :=10000;
       else
         B(I,J).S.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST -
B(I-1,J).CURRENT_COST);
       end if;
       If J = 1 then
          B(I,J).W.WEIGHT := -1;
          B(I,J).W.DISTANCE := -1;
          B(I,J-1).CURRENT_COST :=10000;
       else
        B(I,J).W.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST -
 B(I,J-1).CURRENT_COST);
       end if;
 end CAL_WEIGHT;
------------------------------------------------------------
-- This part finds the minimum cost between current node and
its four
--neighbors(north,east,south,west)
-----------------------------------------------------------
procedure FIND_MIN (I,J : in INTEGER) is
begin
      if B(I,J).CURRENT_COST >  (B(I+1,J).CURRENT_COST
                                 +abs(B(I,J).N.WEIGHT)) then
          B(I,J).CURRENT_COST := (B(I+1,J).CURRENT_COST
                                  +abs(B(I,J).N.WEIGHT));
          B(I,J).DIRECTION    := "NORTH";
       end if;
       if B(I,J).CURRENT_COST >  (B(I,J+1).CURRENT_COST
                                 +abs(B(I,J).E.WEIGHT)) then
          B(I,J).CURRENT_COST := (B(I,J+1).CURRENT_COST
                                  +abs(B(I,J).E.WEIGHT));
          B(I,J).DIRECTION    := "EAST ";
       end if;
       if B(I,J).CURRENT_COST >  (B(I-1,J).CURRENT_COST
                                  +abs(B(I,J).S.WEIGHT)) then
         B(I,J).CURRENT_COST := (B(I-1,J).CURRENT_COST
                                  +abs(B(I,J).S.WEIGHT));
         B(I,J).DIRECTION    := "SOUTH";
      end if;
      if B(I,J).CURRENT_COST >  (B(I,J-1).CURRENT_COST
                                  +abs(B(I,J).W.WEIGHT)) then
         B(I,J).CURRENT_COST := (B(I,J-1).CURRENT_COST
                                  +abs(B(I,J).W.WEIGHT));
         B(I,J).DIRECTION    := "WEST ";
      end if;
end FIND_MIN;
------------------------------------------------------------
begin
   TEXT_IO.OPEN (INF,MODE => TEXT_IO.IN_FILE, NAME =>
"ter.dat");
   TEXT_IO.PUT_LINE("THE DIMENSION OF MATRIX =");
 INTEGER_INOUT.GET(N);
   TEXT_IO.PUT_LINE("THE NUMBER OF PROCESSORS = ");
INTEGER_INOUT.GET(P);
   TEXT_IO.PUT_LINE ("ENTER THE OPTIMIZATION TOLERANCE");
   INTEGER_INOUT.GET (VOLTA);
   TEXT_IO.PUT_LINE ("ENTER THE SOURCE POINT !");
   TEXT_IO.PUT_LINE("SX = ");INTEGER_INOUT.GET (SX);
   TEXT_IO.PUT_LINE("SY = ");INTEGER_INOUT.GET(SY);
   TEXT_IO.PUT_LINE ("ENTER THE GOAL POINT !");
   TEXT_IO.PUT_LINE("GX = ");INTEGER_INOUT.GET (GX);
   TEXT_IO.PUT_LINE("GY = ");INTEGER_INOUT.GET(GY);
------------------------------------------------------------
--This part gets the heights from the file..
------------------------------------------------------------
   for ROW in 1..10 loop
     for COL in 1..10 loop
       INTEGER_INOUT.GET (INF, B(ROW,COL).CURRENT_COST);
     end loop;
   end loop;
------------------------------------------------------------
--It passes the heights to the other processors..
-----------------------------------------------------------
for I in 1..5 loop
  for J in 1 ..10 loop
   GRI := B(I,J);
   DATA_IO.WRITE(OutToMars,GRI);
  end loop;
end loop;
------------------------------------------------------------
--It determines the borders and calculates the weights of the
edges..
------------------------------------------------------------
for I in 6..10 loop
   for J in 1..10 loop
       CAL_WEIGHT(I,J);
   end loop;
 end loop;
-----------------------------------------------------------
--It makes the costs max number in order to use them in
comparisons for finding
-- the minimum....
------------------------------------------------------------
   for I in  6..10 loop
      for J in  1 .. 10  loop
         B(I,J).CURRENT_COST := 10000;
         B(I,J).OLD_COST :=  B(I,J).CURRENT_COST;
      end loop;
   end loop;
------------------------------------------------------------
--This part sends dim of matrix,no of
proccessors,volta,sourceand goal points.
------------------------------------------------------------
GRI.CURRENT_COST:=N;GRI.OLD_COST:=P;GRI.N.WEIGHT:=VOLTA;
GRI.E.WEIGHT:=SX;GRI.S.WEIGHT:=SY;GRI.W.WEIGHT:=GX;GRI.E.DIS
TANCE:=GY;
DATA_IO.WRITE(OutToMars,GRI);
------------------------------------------------------------
--cost minimization...
------------------------------------------------------------
B(SX,SY).CURRENT_COST := 0;
 while COUNTER > 0 loop
    COUNTER := 0;
      for I in 6..10 loop
       for J in 1..10 loop
       if I = 6 then
          GRI := B(I,J);
          DATA_IO.WRITE(OutToMars,GRI);
          DATA_IO.READ(InFromMars,GRI);
          B(I-1,J) := GRI;
      end if;
      FIND_MIN(I,J);
      VOLT := B(I,J).OLD_COST -B(I,J).CURRENT_COST;
      if VOLT > VOLTA then
         COUNTER := COUNTER +1;
      end if;
      B(I,J).OLD_COST := B(I,J).CURRENT_COST;
    end loop;
  end loop;
integer_inout.put(counter);text_io.put("    ");
 GRI.CURRENT_COST := COUNTER;
 DATA_IO.WRITE(OutToMars, GRI);
 DATA_IO.READ(InFromMars, GRI);
 COUNTER := GRI.CURRENT_COST;
 integer_inout.put(counter);
 text_io.put_line(".......................");
end loop;
for I in 1 ..5  loop
  for J in 1 .. 10 loop
    DATA_IO.READ(InFromMars, GRI);
    B(I,J) := GRI;
  end loop;
end loop;
------------------------------------------------------------
--output of least cost path..
------------------------------------------------------------
loop
   PUSH(S,B(GX,GY).CURRENT_COST);
   PUSH1 (S1,B(GX,GY).DIRECTION);
   SQ :=B(GX,GY).DIRECTION;
   if SQ = "NORTH" then
      PUSH(S,B(GX,GY).N.DISTANCE);
      GX := GX+1;
   elsif SQ = "EAST " then
      PUSH(S,B(GX,GY).E.DISTANCE);
      GY := GY+1;
   elsif SQ = "SOUTH" then
     PUSH(S,B(GX,GY).S DISTANCE);
      GX := GX-1;
   elsif SQ =  "WEST " then
      PUSH(S,B(GX,GY).W.DISTANCE);
      GY := GY-1;
   else
   exit;
   end if;
exit when GX =SX and GY =SY ;
end loop;
------------------------------------------------------------
TEXT_IO.PUT_LINE("       DISTANCE      COST   DIRECTION");
TEXT_IO.PUT_LINE("       ---------      ----   ----------");
loop
  POP(S,E);INTEGER_INOUT.PUT(E);TEXT_IO.PUT ("    ");
  POP(S,E);INTEGER_INOUT.PUT(E);TEXT_IO.PUT ("       ");
  POP1(S1,E1);
  if E1 = "NORTH" then
     TEXT_IO.PUT ("SOUTH");TEXT_IO.new_line;
  elsif E1 = "EAST " then
     TEXT_IO.PUT ("WEST");TEXT_IO.new_line;
  elsif E1 = "SOUTH" then
     TEXT_IO.PUT ("NORTH");TEXT_IO.new_line;
  elsif E1 =  "WEST " then
     TEXT_IO.PUT ("EAST");TEXT_IO.new_line;
  end if;
  exit when S.LATEST =0;
end loop;
------------------------------------------------------------
TEXT_IO.CLOSE(INF);
end PROJ0;
```

```
--Title   : PROJ1.ADA
--Author  : CENGIZ EKIN
--Date    : 20/06/92
--Revised : 04/10/92
--Course  : THESIS
--Compiler: MERIDIAN ADA
--Description:
with TEXT_IO, COMMON, CHANNELS;
use COMMON;
procedure PROJ1 is
-- communication channels that are used
OutToEarth  : CHANNELS.CHANNEL_REF := CHANNELS.OUT_PARAMETERS
(2);
InFromEarth : CHANNELS.CHANNEL_REF := CHANNELS.IN_PARAMETERS
(2);
B            : GRID;            GRI  : GRID_POINT;
counter,counter1     : integer := 1;
------------------------------------------------------------
procedure CAL_WEIGHT (I,J : in INTEGER) is
begin
      If I = 10   then
          B(I,J).N.WEIGHT := -1;
          B(I,J).N.DISTANCE := -1;
          B(I+1,J).CURRENT_COST :=10000;
       else
         B(I,J).N.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST -
B(I+1,J).CURRENT_COST);
       end if;
       If J = 10 then
          B(I,J).E.WEIGHT := -1;
          B(I,J).E.DISTANCE := -1;
          B(I,J+1).CURRENT_COST :=10000;
       else
         B(I,J).E.WEIGHT := 1+ABS(B(I,J).CURRENT_COST -
B(I,J+1).CURRENT_COST);
       end if;
       If I = 1 then
          B(I,J).S.WEIGHT := -1;
          B(I,J).S.DISTANCE := -1;
          B(I-1,J).CURRENT_COST :=10000;
       else
         B(I,J).S.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST - B(I-
1,J).CURRENT_COST);
```

```
       end if;
       If J = 1 then
          B(I,J).W.WEIGHT := -1;
          B(I,J).W.DISTANCE := -1;
          B(I,J-1).CURRENT_COST :=10000;
       else
       B(I,J).W.WEIGHT :=1+ ABS(B(I,J).CURRENT_COST - B(I,J-
1).CURRENT_COST);
       end if;
 end CAL_WEIGHT;
-----------------------------------------------------------
procedure FIND_MIN (I,J : in INTEGER) is
begin
      if B(I,J).CURRENT_COST >  (B(I+1,J).CURRENT_COST
                                 +abs(B(I,J).N.WEIGHT)) then
          B(I,J).CURRENT_COST := (B(I+1,J).CURRENT_COST
                                  +abs(B(I,J).N.WEIGHT));
          B(I,J).DIRECTION    := "NORTH";
       end if;
       if B(I,J).CURRENT_COST >  (B(I,J+1).CURRENT_COST
                                 +abs(B(I,J).E.WEIGHT)) then
          B(I,J).CURRENT_COST := (B(I,J+1).CURRENT_COST
                                  +abs(B(I,J).E.WEIGHT));
          B(I,J).DIRECTION    := "EAST ";
       end if;
       if B(I,J).CURRENT_COST >  (B(I-1,J).CURRENT_COST
                                 +abs(B(I,J).S.WEIGHT)) then
         B(I,J).CURRENT_COST := (B(I-1,J).CURRENT_COST
                                 +abs(B(I,J).S.WEIGHT));
         B(I,J).DIRECTION    := "SOUTH";
      end if;
      if B(I,J).CURRENT_COST >  (B(I,J-1).CURRENT_COST
                                 +abs(B(I,J).W.WEIGHT)) then
         B(I,J).CURRENT_COST := (B(I,J-1).CURRENT_COST
                                 +abs(B(I,J).W.WEIGHT));
         B(I,J).DIRECTION    := "WEST ";
      end if;
end FIND_MIN;
-----------------------------------------------------------
begin
for I in 1..5 loop
  for J in 1 ..10 loop
    DATA_IO.READ(InFromEarth, GRI);
    B(I,J) := GRI;
  end loop;
end loop;
------------------------------------------------------------
--It determines the borders and calculates the weights of the
edges..
------------------------------------------------------------
for I in 1 .. 5 loop
   for J in 1..10 loop
       CAL_WEIGHT(I,J);
   end loop;
 end loop;
-----------------------------------------------------------
--It makes the costs max number in order to use them in
comparisons for finding
-- the minimum....
------------------------------------------------------------
   for I in  1 .. 5 loop
      for J in  1 .. 10  loop
         B(I,J).CURRENT_COST := 10000;
         B(I,J).OLD_COST :=  B(I,J).CURRENT_COST;
      end loop;
   end loop;
------------------------------------------------------------
--This part sends dim of matrix,no of
proccessors,volta,sourceand goal points.
------------------------------------------------------------
DATA_IO.READ(InFromEarth,GRI);
N:=GRI.CURRENT_COST;P:=GRI.OLD_COST;VOLTA :=GRI.N.WEIGHT;
SX:=GRI.E.WEIGHT;SY:=GRI.S.WEIGHT;GX:=GRI.W.WEIGHT;GY:=GRI.E
.DISTANCE;
------------------------------------------------------------
--cost minimization...
------------------------------------------------------------
B(SX,SY).CURRENT_COST := 0;
while COUNTER > 0 loop
   COUNTER := 0;
      for I in 1..5 loop
       for J in 1..10 loop
       if I = 5  then
          DATA_IO.READ(InFromEarth,GRI);
          B(I+1,J) := GRI;
          GRI .= B(I,J);
          DATA_IO.WRITE(OutToEarth,GRI);
       end if;
      FIND_MIN(I,J);
      VOLT := B(I,J).OLD_COST -B(I,J).CURRENT_COST;
      if VOLT > VOLTA then
         COUNTER := COUNTER +1;
      end if;
      B(I,J).OLD_COST := B(I,J).CURRENT_COST;
    end loop;
  end loop;
  DATA_IO.READ(InFromEarth, GRI);
  counter1 := GRI.CURRENT_COST;
  if (counter=0) and (counter1=0) then
     GRI.CURRENT_COST := 0;
     DATA_IO.WRITE(OutToEarth, GRI);
  else
     COUNTER :=1;
     GRI.CURRENT_COST :=1;
     DATA_IO.WRITE(OutToEarth, GRI);
  end if;
end loop;
-----------------------------------------------------------
for I in 1..5 loop
  for J in 1 ..10 loop
  GRI := B(I,J);
  DATA_IO.WRITE(OutToEarth,GRI);
  end loop;
end loop;
end PROJ1;
```

```
# File: makefile
#   "make help" to print option list
#
#   Complete development cycle:
#     make family       -- makes Ada family and library
directories
#     make              -- compiles, links, configures source
#     make run          -- run bootable code

MODE = s
PROC = 8
OPTS = /$(MODE) /t$(PROC)

# make the executable code
main.btl: mainh.c$(PROC)$(MODE) proj1h.c$(PROC)$(MODE)
main.pgm
    @ echo EXPECT 1 WARNING...
   iconf /s main.pgm
    @ f:\util\bell

mainh.c$(PROC)$(MODE): proj0.o proj0h.t$(PROC)$(MODE)
merger.t$(PROC)$(MODE) mainh.t$(PROC)$(MODE)
    ilink /f main.lnk

proj0.o: common.ada proj0.ada
    ada invoke proj0.inv,yes

proj0h.t$(PROC)$(MODE): proj0h2.tax proj0h.occ
    occam $(OPTS) proj0h.occ

proj0h2.tax: proj0h2.occ
    occam /ta /x proj0h2.occ

merger.t$(PROC)$(MODE): merger.occ
    occam $(OPTS) merger.occ

mainh.t$(PROC)$(MODE): mainh.occ
    occam $(OPTS) mainh.occ

proj1h.c$(PROC)$(MODE): proj1.o proj1h.t$(PROC)$(MODE)
    ilink proj1h.t$(PROC)$(MODE) proj1.o adarts8.lib
hostio.lib occam8s.lib
```

```
proj1.o: common.ada proj1.ada
    ada invoke proj1.inv,yes

proj1h.t$(PROC)$(MODE): proj1h2.tax proj1h.occ
    occam $(OPTS) proj1h.occ

proj1h2.tax: proj1h2.occ
    occam /ta /x proj1h2.occ
#
# misc.
#
help:
    @ echo Make arguments:
    @ echo   make                   - make from top level down
    @ echo   make -n [opt]          - display but don't execute
commands
    @ echo   make *.o               - make Ada object
    @ echo   make help              - display this list
   @ echo   make clean          - delete all files except source
    @ echo   make run               - run bootable program
    @ echo   make check             - check transputer topology
   @ echo   make family           - make Ada family and library
directories

clean:
    del *.?8?
    del *.tax
    del *.o
    del *.dsc
    del *.btl
    del test_lib\adalib.*
    rd test_lib
    del test_fam\adafam.*
    rd test_fam

run:
    iserver /sb main.btl

check:
    check /r

family:
    ada invoke family.inv,yes
```

```
-- File: main.pgm
#INCLUDE "hostio.inc"
#INCLUDE "linkaddr.inc"
PROTOCOL PASS IS INT;[5]BYTE :

#USE "mainh.c8s"
#USE "proj1h.c8s"

CHAN OF PASS Mars2Earth, Earth2Mars:
CHAN OF SP FromFiler, ToFiler:

PLACED PAR

  PROCESSOR 0 T8

    PLACE FromFiler  AT link0.in:
    PLACE ToFiler    AT link0.out:
    PLACE Mars2Earth AT link2.in:
    PLACE Earth2Mars AT link2.out:

    [1325000] INT ws1:
   main.harness (FromFiler, ToFiler, Mars2Earth, Earth2Mars,
ws1)

  PROCESSOR 1 T8

    PLACE Earth2Mars AT link0.in:
    PLACE Mars2Earth AT link0.out:

    [1280000] INT ws2:
    proj1.harness (Mars2Earth, Earth2Mars, ws2)
```

```
-- File main.lnk
-- Purpose: File list for ilink
mainh.t8s
merger.t8s
hostio.lib
occam8s.lib
( proj0h.t8s proj0.o adarts8.lib hostio.lib occam8s.lib )
```

```
-- File: family.inv
family.new test_fam,overwrite=yes
lib(family=test_fam).new test_lib,overwrite=yes
```

```
-- File: proj0.inv
default.compile library=test_lib
compile common.ada
compile proj0.ada
default.bind library=test_lib,level=bind,warning=no
bind proj0,object="proj0.o",entry_point="proj0.program"
```

```
-- File: proj1.inv
default.compile library=test_lib
compile proj1.ada
default.bind library=test_lib,level=bind,warning=no
bind proj1,object="proj1.o",entry_point="proj1.program"
```

```
-- File: mainh.occ
#OPTION "AGNVW"
#INCLUDE "hostio.inc"
PROTOCOL PASS IS INT;[5]BYTE :
PROC main.harness (CHAN OF SP  FromFiler, ToFiler,
                   CHAN OF PASS Mars2Earth, Earth2Mars,
                    []INT FreeMemory)

  #USE "hostio.lib"

  #USE "proj0h.t8s"
  #USE "merger.t8s"

  [1]CHAN OF ANY Debug:
  [2]CHAN OF SP FromAda, ToAda:
  CHAN OF BOOL StopDebug, StopMultiplexor:
  SEQ

    PAR

     -- A multiplexor to combine the debug and normal output.
      so.multiplexor (FromFiler, ToFiler, FromAda, ToAda,
StopMultiplexor)

      -- A debug channel merger.
      debug.merger (ToAda[0], FromAda[0], Debug, StopDebug)

      -- A process to invoke the sieve program.
      ws IS FreeMemory:
      SEQ
        proj0.harness (FromAda[1], ToAda[1], Debug[0],
Mars2Earth, Earth2Mars, ws)
        StopDebug ! FALSE
        StopMultiplexor ! FALSE

    so.exit (FromFiler, ToFiler, sps.success)
:
```

```
-- File: merger.occ

#OPTION "AGNVW"
#INCLUDE "hostio.inc"

PROC debug.merger (CHAN OF SP FromFiler, ToFiler,
                   []CHAN OF ANY Debug,
                   CHAN OF BOOL Stop)

  #USE "hostio.lib"

  -- A debug channel merger and blocker.

  VAL max.debug IS 20:
  VAL number.of.debug IS SIZE Debug:

  INT line.index:
  [256]BYTE line.buffer:
  BYTE value, r:
  BOOL running, reset, s:
  [max.debug]BOOL mask:
  VAL BYTE line.feed IS 10 (BYTE):
  SEQ
    SEQ i = 0 FOR number.of.debug
      mask[i] := TRUE
    running := TRUE
    reset := FALSE
    line.index := 0
    WHILE running
      PRI ALT
        ALT i = 0 FOR number.of.debug
          mask[i] & Debug[i] ? value
            SEQ
              IF
                value = line.feed
                  SEQ
                    -- Send the complete line.
                   so.puts (FromFiler, ToFiler, spid.stdout,
                     [line.buffer FROM 0 FOR line.index], r)
                    line.index := 0
                    mask [i] := FALSE
                    reset := TRUE
                TRUE
```

```
                  SEQ
                    -- Add character to line.
                    line.buffer[line.index] := value
                    line.index := line.index + 1
        reset & SKIP
          SEQ
            reset := FALSE
            SEQ i = 0 FOR number.of.debug
              mask[i] := TRUE
        Stop ? s
          running := FALSE
:
```

```
-- File: proj0h.occ

#OPTION "AGNVW"
#INCLUDE "hostio.inc"
PROTOCOL PASS IS INT;[5]BYTE :
PROC proj0.harness (CHAN OF SP  FromAda, ToAda,
                    CHAN OF ANY Debug,
                    CHAN OF PASS Mars2Earth, Earth2Mars,
                    []INT FreeMemory)

  #IMPORT "proj0h2.tax"

  [1]INT dummy.ws:
  ws1 IS FreeMemory:
  [3]INT in.program:
  [3]INT out.program:
  SEQ
    -- Set up vector of pointers to channels.
    in.program[0] := MOSTNEG INT   -- not used
    LOAD.INPUT.CHANNEL (in.program[1], ToAda)
    LOAD.INPUT.CHANNEL (in.program[2], Mars2Earth)
    LOAD.OUTPUT.CHANNEL (out.program[0], Debug)
    LOAD.OUTPUT.CHANNEL (out.program[1], FromAda)
    LOAD.OUTPUT.CHANNEL (out.program[2], Earth2Mars)
    -- Invoke the Ada program.
    -- Assumes the entry point name has been changed to
"proj0.program".
    proj0.program (ws1, in.program, out.program, dummy.ws)
:

-- File: PROJ0h2.occ
#OPTION "AEV"
PROC proj0.program ([]INT ws1, in, out, ws2)
  [1000]INT d:
  SEQ
    SKIP
:
```

```
-- File: proj1h.occ

#OPTION "AGNVW"
#INCLUDE "hostio.inc"
PROTOCOL PASS IS INT;[5]BYTE :

PROC proj1.harness (CHAN OF PASS Mars2Earth, Earth2Mars,
                     []INT FreeMemory)

  #IMPORT "proj1h2.tax"

  [1]INT dummy.ws:
  ws1 IS FreeMemory:
  [3]INT in.program:
  [3]INT out.program:
  SEQ
    -- Set up vector of pointers to channels.
    in.program[0] := MOSTNEG INT     -- not used
    in.program[1] := MOSTNEG INT     -- standard i/o not used
    LOAD.INPUT.CHANNEL (in.program[2], Earth2Mars)
    out.program[0] := MOSTNEG INT    -- standard i/o not used
    out.program[1] := MOSTNEG INT    -- standard i/o not used
    LOAD.OUTPUT.CHANNEL (out.program[2], Mars2Earth)
    -- Invoke the Ada program.
    -- Assumes the entry point name has been changed to
"proj1.program".
    proj1.program (ws1, in.program, out.program, dummy.ws)
:
```

```
-- File: proj1h2.occ

#OPTION "AEV"

PROC proj1.program ([]INT ws1, in, out, ws2)
  [100000]INT d:
  SEQ
    SKIP
:
```

INITIAL DISTRIBUTION LIST

Defense Technical Information Center 2
Cameron Station
Alexandria, VA 22304-6145

Library, Code 52 2
Naval Postgraduate School
Monterey, CA 93943-5002

Chairman, Code CS 2
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5100

Professor Amr Zaky 1
Code CS/Za
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5100

Professor Mantak Shing 1
Code CS/Sh
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5100

Professor Robert McGhee 1
Code CS/Mz
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5100

Professor Shridhar Shukla 1
Code EC/Sh
Department of Electrical & Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5100

Professor Anthony Healey 1
Code ME/Hy
Department of Mechanical Engineering
Naval Postgraduate School
Monterey, CA 93943-5100

Professor Sehung Kwak 1
Code CS/Kw
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5100

Deniz Kuvvetleri Komutanligi 1
Personel Daire Baskanligi
Bakanliklar, Ankara / TURKEY

Golcuk Tersanesi Komutanligi 2
41650 Golcuk, Kocaeli / TURKEY

Deniz Harp Okulu Komutanligi 2
81704 Tuzla, Istanbul / TURKEY

Taskizak Tersanesi Komutanligi 2
Kasimpasa, Istanbul / TURKEY

Cengiz EKIN 2
Gulistan Sok. No:13-6
Abidinpasa Ankara / TURKEY